Aprendendo a
PROGRAMAR
LINGUAGEM JAVA
Tito Petri
JAVA

1. O Que Aprenderemos neste Livro?

Neste material você vai aprender sobre **Programação de *Softwares***, começando do zero, até criar suas **7 Primeiras Aplicações** com a **Linguagem Java**.

O objetivo deste Livro é fazer com que o aluno aprenda a parte teórica da **Lógica de Programação**, conhecendo os principais termos, conceitos, e a técnica de se criar algoritmos e programas de computador.

Quando falamos sobre **Códigos de Programação**, muitos pensam que a parte teórica não é muito importante e acabam apenas decorando ou copiando códigos.

Através deste Livro, vou te ensinar que programar é o mesmo que criar um método para **solucionar algum problema**. Esta é uma grande habilidade de raciocínio que você levará para sua vida.

Ao final deste material, você terá adquirido novos conhecimentos em:

- Lógica de Programação;
- Criação de Algoritmos;
- Linguagem de Programação Java;
- Criação do seu Primeiro Aplicativo Android.

Diga "Olá!" para o **Felpudo**!

O Passarinho Desenvolvedor mais *Geek*, *Nerd e Intelectual* que você jamais conheceu!

Felpudo

O Felpudo vai nos ensinar tudo sobre programação, passando as idéias e conceitos de forma ilustrada, lúdica e divertida.

Este livro busca trazer uma nova e mais prazerosa experiência de aprendizado, de modo que, tanto um jovem quanto uma criança sejam capazes de aprender a

programar, e também de desenvolver o gosto pelo estudo e pela prática.

Criaremos **7 Projetos** desafiadores para treinar a técnica de desenvolver programas com a Linguagem Java: desde o conceito inicial, até a aplicação funcionando.

Para isso, utilizaremos apenas **ferramentas gratuitas** que funcionam dentro do seu navegador (*browser)*, sem precisar comprar, baixar, ou instalar nenhum programa adicional. Você só vai precisar de um **computador** e **acesso à *Internet***.

Ao final do material, também vou te ensinar a **instalar** e dar os **Primeiros Passos** no **Android Studio IDE**, para que você crie seu **Primeiro Aplicativo Android** utilizando o Java.

O conhecimento adquirido com este material, vai te dar um ótimo embasamento para aprender qualquer outra linguagem de programação como **C++, *Swift***, ***C#***, ***Python*** ou ***JavaScript***.

Programar é uma valiosa habilidade que, nos próximos anos, se aplicará em praticamente todos os campos e áreas de atuação.

Lembre-se de que os conhecimentos em Lógica e Algoritmo são aplicados ao desenvolvimento de **todo tipo** de **Aplicação Digital**, como em **VideoGames**, **Criação de Sites**, **Sistemas**, **Produtos Eletrônicos**, **Robótica**, e outros!

2. Linguagem de Programação Java

Java é hoje uma das linguagens de programação mais populares do mundo. A maioria dos dispositivos (computadores, celulares e tablets) suportam aplicações desenvolvidas em Java.

Acompanhando este material, você vai aprender rápida e metodicamente tudo sobre o fundamento da **Lógica de Programação**, já aplicada à linguagem ***Java***.

Vamos criar juntos **7 Programas Completos em Java,** e iremos aprender simultaneamente Lógica de Programação e, consequentemente, utilizaremos a linguagem Java para tanto. Veja abaixo os algoritmos que iremos desenvolver:

1. Par ou Ímpar;
2. Jokempô;
3. Sorteio de Nomes;
4. Buscar Nome;
5. Validação do CPF;
6. Números Primos;
7. Criar e Consultar Cadastro.

3. Compilador Online de Java

O Java é uma linguagem utilizada em vários programas, compiladores e IDEs. Existem muitas maneiras diferentes de programar em Java.

Cada ferramenta vai te possibilitar a criação de um tipo diferente de aplicação (games, aplicativos *mobile*, aplicativos *desktop)*.

Neste livro utilizaremos um **Compilador Online de Java**. Uma ferramenta **gratuita** que serve para estudar e prototipar programas e algoritmos.

Os **Compiladores Online** são ferramentas bem simples de utilizar, e não necessitam de nenhuma instalação, pois funcionam diretamente dentro do navegador da *internet*.

São ferramentas um pouco limitadas em matéria de recursos, porém são ideais para estudos e criação de protótipos de programações.

Ao final do material, vou te ensinar a utilizar o Android Studio IDE (*Integrated Development Environment*) junto com o Java, para que você inicie no desenvolvimento de aplicações *mobile*.

O Android Studio é o programa capaz de juntar o Java com qualquer outro tipo de recurso Gráfico (Imagens, Animações, Vídeos) e de funcionalidades (*Internet*, Banco de Dados e todo tipo de recurso dos celulares e tablets Android como Internet, Câmera, Acelerômetro ou Geolocalização).

Porém, o IDE requer muito mais conhecimento para ser instalado e utilizado. Recomendo que o leitor ou aluno sempre inicie por um compilador mais simples para entender os fundamentos da programação, antes de começar a estudar o IDE. Isto vai facilitar muito o aprendizado.

Seguem algumas sugestões de **Compiladores Java Online Gratuitos**:

- www.compilejava.net
- www.tutorialspoint.com/compile_java_online.php
- www.browxy.com
- www.codiva.io
- www.jdoodle.com
- ideone.com
- www.onlinegdb.com
- rextester.com/l/java_online_compiler

Para as explicações do Livro, utilizarei o **compilejava.net.** Porém, todos os outros compiladores têm funções e recursos bem parecidos, e também podem ser utilizados durante o desenvolvimento dos projetos.

Então vamos começar! Acesse o site ***compilejava.net*** através do seu navegador.

Observe a interface do programa e vamos entender como utilizar o **Compilador Java**.

Precisamos conhecer basicamente **4 recursos** da ferramenta:

1. Área para Escrever o Código
2. Botão para Compilar e Executar o Código
3. Console de Status e Erros de Execução
4. Console para Exibir Mensagens

Observe o código que já vem escrito como exemplo. Clique em **COMPILE & EXECUTE (2)** e veja o que aparece no **Console de Mensagens (3)**.

Esta mensagem é o resultado da aplicação que foi executada. Em cada um dos sites/compiladores, você vai ter um exemplo diferente. Não vamos nos aprofundar nisso agora.

Apague todo o código e vamos escrever um programa do Zero!

Copie o código a seguir **exatamente como está,** e execute no seu compilador para ver o que acontece.

Lembre-se que cada **vírgula**, **parênteses** ou **chaves** são importantes e fazem toda a diferença!

Exemplo:

```
public class MinhaClasse
{
  public static void main( String[ ] args )
    {
      System.out.print( "Bem-Vindo ao Java!" );
    }
}
```

Clique em **COMPILE & EXECUTE** e veja o que aparece agora no **Console de Mensagens**.

Resultado:

```
Bem-Vindo ao Java!
```

O resultado é praticamente o mesmo que anteriormente, apenas uma mensagem sendo exibida no console. Porém agora utilizamos apenas o **essencial**, o **mínimo de código** para fazer um código em Java funcionar, ou seja:

- uma **Classe** (no caso, apelidei de ***MinhaClasse***);
- e uma **Função** chamada ***main*** dentro desta classe;

* Para reforçar o que você acabou de aprender neste capítulo, assista esta videoaula sobre o mesmo assunto no meu canal do YouTube: https://youtu.be/Fftgf2vUoYc

4. Regras Básicas para Programar

Primeiro precisamos entender algumas regras básicas do Java, e de programação em geral para entender 100% do que está escrito neste código. Comece memorizando estas informações:

a) Case Sensitive

O Java é ***Case Sensitive***, ou seja, ele leva em consideração a diferença entre letras maiúsculas e letras minúsculas. Veja o seguinte comando para exibir um valor no console.

Exemplo:

```
System.out.print()
```

Esse comando nunca poderá ser escrito como:
System.Out.Print() ou **system.out.print()** ou **SYSTEM.OUT.PRINT(),** pois isso ocasionará erros em sua aplicação.

b) Chaves

As Chaves (**{ }**) indicam uma estrutura. A Classe, por exemplo, é uma estrutura que começa com **{** e termina com **}**. Isso também acontece com a Função **main**.

c) Aspas Duplas

Para escrever uma **palavra**, **frase** ou **cadeia de caracteres** (letras) devemos iniciar e terminar sempre com **aspas duplas** (**"**).

Agora, baseando-se nos conhecimentos adquiridos até agora, leia atenciosamente o comando a seguir, e perceba quantos conceitos estão envolvidos em uma única linha de código:

Exemplo:

```
System.out.print( "Bem-Vindo ao Java!" );
```

Vamos entender palavra por palavra o que diz o nosso código:

O comando **System.out.print()** serve para exibir uma mensagem no console.

Exemplo:

```
System.out.print( "Bem-Vindo ao Java!" );
```

Entre os parênteses, passamos o argumento para este comando. Ou seja, o objeto ou valor que deve ser exibido.

Exemplo:

```
System.out.print( "Bem-Vindo ao Java!" );
```

Esse argumento será, neste caso, uma frase, ou uma cadeia de caracteres (ao qual chamamos de *String*), que então se encontrará , necessariamente, entre aspas duplas.

Para indicar o fim da linha de comando, utilizamos o **ponto e vírgula** (**;**) que a finaliza .

Exemplo:

```
System.out.print( "Bem-Vindo ao Java!" );
```

Viu!? Cada caractere, letra, número, vírgula ou ponto tem um porquê na programação.

Ao criar um código, você deve saber exatamente o que cada letrinha está querendo dizer.

Agora, solucione estes **Desafios** para provar o que você aprendeu:

DESAFIO #1

Eu te desafio a fazer um Programa em Java que imprime o seu nome completo no Console de Mensagens.

Faça-o da maneira mais simples, utilizando o mínimo possível de linhas de código.

Lesmo

DESAFIO #2

Bugado

Pesquise sobre as Origens da Linguagem Java.

Quando e onde ela foi criada?

Qual foi a empresa ou programador(es) que a criaram?

Mas atenção: antes de passar para a próxima fase, certifique-se que aprendeu e experimentou de verdade todos os conhecimentos mencionados nos desafios acima.

Você terá que utilizá-los o tempo todo!

Compartilhe este conhecimento nas suas redes sociais, e mostre aos seus amigos o que você aprendeu de novo!

d) Ponto e Vírgula

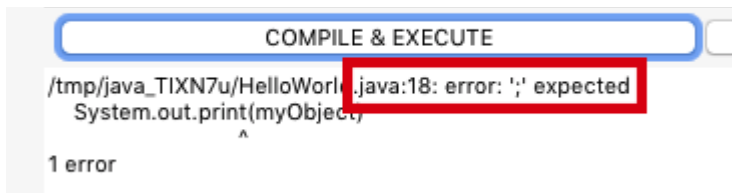
O ponto e vírgula indica ao compilador que o comando chegou ao fim.

No Java e em várias outras linguagens de programação, você precisa indicar o fim da linha de comando.

Lembrem-se que os **espaços** (caracteres vazios), **parágrafos** e **linhas em branco** são desconsiderados pelo compilador.

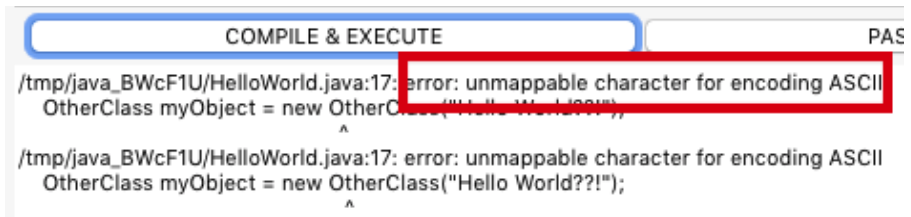
Na **ausência do ponto e vírgula**, o compilador geralmente gera um erro no console:

COMPILE & EXECUTE

```
/tmp/java_TIXN7u/HelloWorld.java:18: error: ';' expected
   System.out.print(myObject)
                             ^
1 error
```

É importante saber interpretar os erros do compilador! Este erro ocorre na linha (...**18: error: ';' expected**...).

e) Caracteres com Acentuação

Alguns compiladores online não funcionam com determinados **Caracteres ASCII da Língua Portuguesa** (caracteres com acento, principalmente).

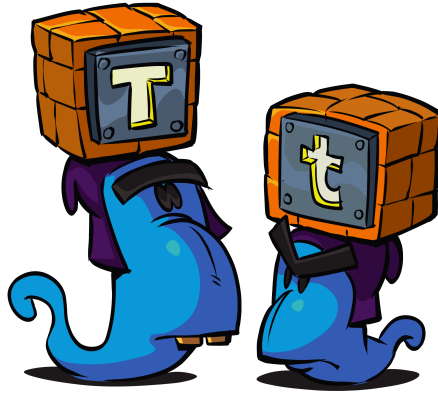
Se encontrar o seguinte erro, por exemplo, perceba que pode estar usando algum acento ou caractere não reconhecido pelo sistema:

OBSERVAÇÃO:
Como os caracteres acentuados não funcionam em alguns compiladores, afim de não confundir alguns leitores durante a exemplificação dos códigos, **nunca usarei acentuação** nas caixas de exemplo! Experimente sempre se seu compilador aceita ou não acentuação em português.

f) Case Sensitive

Vou reforçar aqui novamente, que a Linguagem Java é Case Sensitive, ou seja, trata os caracteres **maiúsculos** e **minúsculos** diferenciadamente.

Lembre-se que no Java, uma variável chamada **nome** é diferente de **NOME** ou **Nome**.

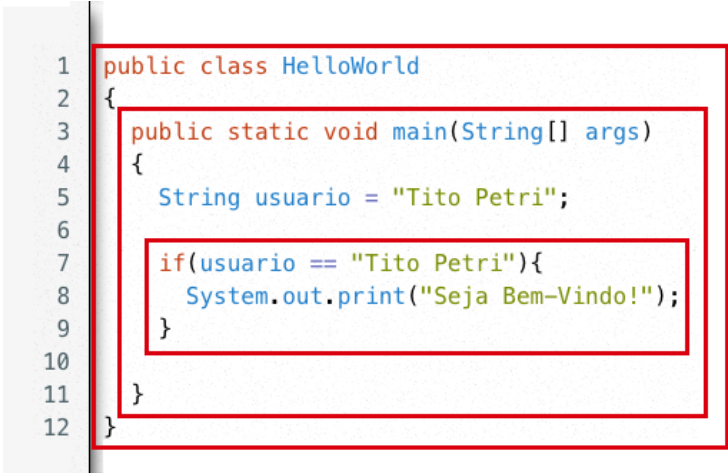
g) Camel Case

É a prática de escrever palavras ou frase compostas de modo que cada palavra comece com uma letra maiúscula, sem espaços ou pontuação intermediários.

Isto facilita a leitura do código, e tornou-se um padrão entre os bons programadores. Observe:

eusouumavariavel ou **EuSouUmaVariavel**

Perceba que no "Caso do Camelo" a frase fica muito mais legível. Exemplos comuns incluem "iPhone" e "eBay".

Então, lembre-se de usar esta boa prática na hora de criar os nomes para suas **classes**, **funções** e **variáveis** na programação.

Outra regra que vale lembrar, é que existem o **lowerCamelCase**, onde a primeira letra inicia em

minúsculo (geralmente usamos para declarar **variáveis** ou **métodos**), e o **UpperCamelCase** com a primeira iniciada em **maiúsculo** (usada geralmente para declarar **classes**).

Mas não se preocupe, pois entenderemos adiante o que são estas **variáveis**, **métodos** e **classes**.

h) Limites ou Escopos

Escopo significa um local bem determinado com o intento de se atingir. Significa o limite ou a abrangência de uma operação.

Na programação, definimos os **escopos** ou **limites** de um **método** ou **classe** utilizando o caractere das **chaves {...}**

```
1  public class HelloWorld
2  {
3    public static void main(String[] args)
4    {
5      String usuario = "Tito Petri";
6
7      if(usuario == "Tito Petri"){
8        System.out.print("Seja Bem-Vindo!");
9      }
10
11   }
12 }
```

Por enquanto, apenas perceba que no trecho desse código existem três objetos, um contido no outro, sendo que, estes 3 objetos são definidos ao se **abrir e fechar as chaves**.

Através deste concepção de escopos, estruturamos a nossa programação em um conceito de **hierarquia** ou **parentesco,** que também define a **visibilidade** e o **acesso** à estes objetos.

Por enquanto, entenda basicamente que, quando você abre e fecha uma chave na programação, você está criando esses blocos de objetos.

E lembre-se de que sempre que abrir uma chave, você deverá fechá-la!

5. Entenda bem este Material

Antes de sair escrevendo as linhas de código, é muito importante entender alguns conceitos fundamentais da programação em geral.

Preste atenção nestes pontos para conseguir entender e absorver bem este material.

a) Quadro de Exemplos de Código

Todos os códigos que aparecem pela primeira vez, estarão destacados em **negrito** dentro do quadro de exemplo.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[ ] args)
  {
      System.out.print("Hello World!");
  }
}
```

Os códigos que aparecerão no quadro de exemplo procuram mostrar apenas os novos comandos, que dificilmente vão se repetir.
Já os comandos referentes ao assunto que estamos abordando, aparecerão sempre em **negrito**.

b) Códigos e projetos para baixar

Todos os códigos e projetos desenvolvidos no decorrer deste Livro, estarão disponíveis através do link abaixo:

http://titopetri.com/recursos/CodigosCursoJava.zip

c) Curso Online de Android com Java

Este Livro também foi desenvolvido em formato de vídeo aulas online. Caso você queira, também pode assistir as video aulas enquanto acompanha este Livro.

Acesse a página do curso online de programação em Java, e aprenda a programar junto com a leitura deste livro. Assim, você pode utilizar esse material como um bom apoio.

6. Variáveis no Java

Para programar, precisamos manipular valores ou dados que são armazenados em "recipientes" identificados.

Imagine uma caixa com algum objeto guardado nela. Esta caixa possui um nome para você se lembrar o que ela contém.

É claro que a caixa não existe, ela representa os **espaços na memória** do dispositivo.

Para o computador, esta variável tem um endereço numérico que indica a posição na memória.

Na nossa programação, vamos dar nomes às variáveis, que representam estes endereços.

Pense que a Memória RAM do computador é um armário, separado em vários recipientes, cada um com um nome (Variável), sendo que, em cada recipiente você tem a possibilidade de guardar um determinado tipo de item (Dado ou Valor).

Veja abaixo alguns dos principais **Tipos de Dados** que podemos criar e utilizar na programação em geral.

TIPO	DESCRIÇÃO	EXEMPLO
boolean	Verdadeiro ou Falso	true, false
int	Números Inteiros	7, -10, 500
float	Números Decimais	1.5f, 0.75f, -5.5f
double	Números Decimais	100.0, 5.7, 1.0, 3.2
char	Único Caractere	\`a\`,\`b\`,\`c\`,\`x\`,\`7\`,\`!\`
String	Palavras e Frases	"Tito","Felpudo","Fofura"

Cada variável só pode ser de um único tipo, e só pode armazenar objetos daquele mesmo tipo.

Por exemplo: você nunca poderá guardar um nome em uma variável que já foi numérica.

Para criarmos uma variável na programação em Java, precisamos seguir uma regra.

Veja o exemplo da criação de uma variável do tipo String, para armazenar um nome:

Exemplo:

```
String nome = "Tito Petri";
```

Perceba que precisamos seguir uma lógica para dizer ao compilador: que **tipo de dado** estamos criando, **qual o nome** desta variável, e a sua **inicialização**, que é o valor que será guardado na variável. Desse modo:

1) Declare o **Tipo** da variável (**String**)
2) Declare um nome para a variável (**nome**)
3) Inicie-a com algum valor (**=** "Tito Petri")
4) Lembre-se do ponto e vírgula para indicar o fim do comando (**;**)

Neste momento da **Declaração das Variáveis,** precisamos entender mais algumas regras:

1) Podemos usar qualquer nome para as variáveis. Contando que o nome respeite o seguinte:

Não seja uma **Palavra Reservada** (comando da linguagem). (Exemplo: System, class, void…)

Não pode conter um **Caractere Especial**.

(Exemplo: ! @ #$%ˆ&*)

- Não pode **Iniciar em Número**.
(Exemplo: 50nome, 123, 1idade)

2) É possível declarar apenas a variável sem inicializá-la.

Escreva o seu **Nome** e **Tipo**) e atribua à ela um valor apenas adiante no código.

Exemplo:

```
String nome;

nome = "Tito Petri e Felpudo";
```

3) O símbolo de igual (**=**) neste momento não tem o sentido de **comparação,** e sim de **atribuição**. Ou seja, estamos atribuindo, ou guardando um valor em uma determinada variável.

Exemplo:

```
int idade =  12;
```

No exemplo acima, criamos uma variável do tipo **int**, damos a ela o nome de **idade,** e guardamos nela o valor **12**.

4) Os números decimais são representados por dois tipos diferentes (**float** ou **double**).

A diferença entre estes dois tipos, é que o **double** armazena o dobro de casas decimais depois do ponto (o que garante uma maior precisão do número, e é ideal para aplicações gráficas)

Consequentemente, o **float** é mais leve e ocupa menos espaço na memória.

Um **float** deve sempre acompanhar um **f** após o número.

Veja a seguir o exemplo da declaração de diferentes tipos de variáveis dentro do escopo (chaves **{...}**) da função **main**:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    String nome = "Tito Petri";
    char sexo = 'm';
    int idade = 23;
    float altura = 1.85f;
    double peso = 70.5;
    boolean ligado = true;
  }
}
```

Declare algumas variáveis e execute o código.

Por enquanto, se o seu código estiver todo correto, nada irá acontecer, pois apenas criamos algumas variáveis e atribuímos alguns valores à elas.

Agora, vamos acessá-las e exibir seu valor no console. Para isso, vamos utilizar o comando **System.out.print()**.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[ ] args)
  {
    String nome = "Tito Petri";
    int idade = 23;
    float altura = 1.85f;
    boolean ligado = true;

    System.out.print(nome);
    System.out.print(idade);
    System.out.print(altura);
    System.out.print(ligado);
  }
}
```

Execute o código e veja o resultado que aparece no seu Console de Mensagens.

Resultado:

```
Tito Petri231.85true
```

Perceba que os valores foram impressos um em sequência do outro, sem espaço.

Para executar uma quebra de linha entre um dado e outro, troque o comando **print** por **println**

Exemplo:

```
System.out.println(nome);
System.out.println(idade);
System.out.println(altura);
System.out.println(ligado);
```

Resultado:

```
Tito Petri
23
1.85
true
```

* Para reforçar o que você acabou de aprender neste capítulo, assista esta videoaula sobre o mesmo assunto no meu canal do YouTube: https://youtu.be/XXSz2FGvUBA

7. Comentando o Código

Existe a possibilidade de comentar linhas ou pedaços do código que você está escrevendo.

O Compilador, nesse caso, irá ignorar a parte comentada na compilação e na execução do programa.

No Java, portanto, utilizamos os seguintes operadores:

//	Comenta a linha atual.
/*	Inicia um comentário em múltiplas linhas.
***/**	Encerra um comentário em múltiplas linhas.

Os comentários servem para você **descrever** e **documentar** um determinado trecho do código ou apenas **omitir algum comando**.

Esta documentação vale tanto para você mesmo entender o que está programando, quanto para outros programadores que eventualmente venham a usar o seu código ou projeto.

Exemplo:

```
/*
   Código criado por Tito Petri
   Este aqui é um comentário
   feito em múltiplas linhas
*/

public class HelloWorld
{
  public static void main(String[] args)
  {
    // Declaração das Variáveis

    String nome = "Tito Petri";
    int idade = 23;
    float altura = 1.85f;

     // Exibindo as Variáveis no Console

    System.out.println(nome);
    System.out.println(idade);
    System.out.println(altura);
  }
}
```

8. Operações com Strings

O tipo **String** se refere à uma palavra, frase ou qualquer cadeia de vários caracteres (ou letras).

Existem algumas operações (ou métodos) que podemos utilizar com este tipo de objeto. Confira alguns:

a) toUpperCase()

converte todos os caracteres em maiúsculo.

b) length()

Nos informa o tamanho do String, ou quantas letras (caracteres) ele tem.

c) equals

Faz a comparação entre 2 Strings. Retorna *true* quando são iguais.

d) replace('a', 'b')

Substitui o caractere do argumento 1 (parâmetro antes da vírgula), pelo argumento 2 (depois da vírgula). Repare

que este método vale apenas para **um caractere,** e por isso utilizamos **aspas únicas**.

e) replaceFirst("abc","xyz")

Substitui os caracteres do primeiro argumento, pelo segundo, apenas na primeira ocorrência encontrada.

f) replaceAll("abc","xyz")

Substitui a ocorrência do primeiro argumento, pelo segundo, em todas as ocorrências encontradas no String.

g) Character to String

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    String nome = "Felpudinho";
    for (int i = 0; i < nome.length(); i++) {
      char c = nome.charAt(i);
      String msg = Character.toString(c);
      String MSG = msg.toUpperCase();
      System.out.print(MSG+".");
    }
  }
}
```

Resultado:

```
F.E.L.P.U.D.I.N.H.O.
```

```
1  public class HelloWorld
2  {
3    public static void main(String[] args)
4    {
5      String nome = "Felpudinho";
6      System.out.print(nome.charAt(3));
7    }
8  }
```

(command line arguments)

COMPILE & EXECUTE | PASTE SOURCE | DOWNLOAD ZIP | sc

Successfully compiled /tmp/java_pt2gki/HelloWorld.java <-- main method

p

```
1  public class HelloWorld
2  {
3    public static void main(String[] args)
4    {
5      String nome = "Felpudinho";
6      for (int i = 0; i < nome.length(); i++) {
7        System.out.print(nome.charAt(i)+".");
8      }
9    }
10 }
11
```

(command line arguments)

COMPILE & EXECUTE | PASTE SOURCE | DOWNLOAD ZIP | solari

Successfully compiled /tmp/java_ZKfPOB/HelloWorld.java <-- main method

F.e.l.p.u.d.i.n.h.o.

```
1  public class HelloWorld
2  {
3    public static void main(String[] args)
4    {
5      String nome = "Felpudinho";
6      for (int i = 0; i < nome.length(); i++) {
7        char c = nome.charAt(i);
8        String msg = Character.toString(c);
9        String MSG = msg.toUpperCase();
10       System.out.print(MSG+".");
11     }
12   }
13 }
```

(command line arguments)

COMPILE & EXECUTE | PASTE SOURCE | DOWNLOAD ZIP

Successfully compiled /tmp/java_x6z3XI/HelloWorld.java <-- main method

F.E.L.P.U.D.I.N.H.O.

Repare que estes métodos são utilizados especificamente em **Strings**. Para utilizá-los, vamos referenciar o **nome do String** e utilizar o **ponto (.)** seguido do método.

Observe a chamada de cada um dos **Métodos de String** mencionados e seus respectivos resultados.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    String nome = "Oi eu sou o Tito Petri!";

    System.out.println( nome );
    System.out.println( nome.toUpperCase() );
    System.out.println( nome.length() );
    System.out.println( nome.equals("Tito") );
    System.out.println( nome.replace('o', 'x') );
    System.out.println( nome.replaceFirst("t", "yyy") );
    System.out.println( nome.replaceAll("o", "zzz") );
  }
}
```

Resultado:

```
Oi eu sou o Tito Petri!
OI EU SOU O TITO PETRI!
23
false
Oi eu sxu x Titx Petri!
Oi eu sou o Tiyyyo Petri!
Oi eu szzzu zzz Titzzz Petri!
```

9. Comandos Unicode para Strings

Ao utilizar um **String** podemos usar operadores ***Unicode*** (Padrão Internacional de Caracteres), que nos auxiliam a montar e formatar textos para exibir mensagens no console. Por exemplo:

COMANDO UNICODE	DESCRIÇÃO
\t	Insere um parágrafo
\n	Quebra de Linha
\"	Aspas Duplas
\'	Aspas SImples
****	Barra Invertida

Execute no seu compilador alguns comandos Unicode para testar a formatação dos Strings.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println("\tParagrafo");
    System.out.println("\nQuebra Linha");
    System.out.println("\"Aspas Duplas\"");
    System.out.println("\'Aspas Simples\'");
    System.out.println("\\Barra Invertida");
```

```
  }
}
```

Resultado:

```
            Paragrafo

Quebra Linha
"Aspas Duplas"
'Aspas Simples'
\Barra Invertida
```

Perceba nos últimos 3 exemplos que a **barra invertida** (**\\ )** dentro de uma cadeia de caracteres, faz o compilador **ignorar o próximo caractere** e com isso conseguimos inserir uma barra invertida ou aspas dentro do **String**.

10. Convertendo Tipos de Variáveis

Em determinados momentos na programação, será necessário manipular os **Tipos das Variáveis** para chegarmos em determinados resultados.

Adiante, neste material, iremos nos deparar com situações como esta. Por isso, vamos entender primeiro as operações para **Converter os Tipos de Variáveis**.

Veja alguns exemplos de conversão de tipos:

a) **String** para **int**

Exemplo:

```
String palavra = "500";
int numero = Integer.parseInt(palavra);
```

b) **int** para **String**

Exemplo:

```
int numero = 10;
String palavra = String.valueOf(numero);
```

Ou então outra maneira ainda mais simples:

Exemplo:

```
int numero = 10;
String numero = " " + numero;
```

c) Qualquer Tipo para String

Basta **somar** a variável à um **String** (praticamente o mesmo exemplo feito anteriormente).

Veja que no final todas as variáveis são unidas (concatenadas) em um único **String.**

Exemplo:

```
int idade = 13;
String nome = "Felpudo tem " + idade + " anos de vida.";

System.out.print( nome );
```

d) float para int

```
float decimal = 17.05f;
int inteiro = (int)decimal;
```

11. Operações Matemáticas

Um recurso indispensável para a programação, é o de fazer facilmente **Operações Matemáticas** pelo Compilador.

As **4 Operações Fundamentais** são realizadas facilmente pelos operadores:

+	Soma
-	Subtração
*****	Multiplicação
/	Divisão

Podemos executar tanto as **Operações Fundamentais da Aritmética**, quando trabalhar com expressões e funções trigonométricas mais avançadas.

Além das 4 operações básicas, confira algumas outras operações matemáticas que conseguimos realizar no Java:

+ (addition)
++ (increment)
+= (add assign)

- (minus)
-- (decrement)
-= (subtract assign)

* (multiply)
*= (multiply assign)

/ (divide)
/= (divide assign)

% (modulo)

a) Módulo

O módulo é calculado com o operador **%**. Ele nos retorna o **resto** entre a divisão de dois números.

Exemplo:

```
System.out.println( 7%3 );
System.out.println( 8%2 );
System.out.println( 4%5 );
```

Resultado:

```
1
0
4
```

b) Raiz Quadrada - Math.sqrt()

O comando **Math** nos permite realizar operações ainda mais avançadas como o **cálculo da raiz** ou da **potência** de um número. Veja os exemplos a seguir:

Exemplo:

```
System.out.println( Math.sqrt(100) );
System.out.println( Math.sqrt(4) );
System.out.println( Math.sqrt(121) );
```

Resultado:

```
10.0
2.0
11.0
```

c) Potência - Math.pow()

Exemplo:

```
System.out.println( Math.pow(2, 2) );
System.out.println( Math.pow(4, 2) );
System.out.println( Math.pow(3, 3) );
```

Resultado:

```
4.0
16.0
27.0
```

12. Arredondamento de Decimais

Conheça também alguns métodos para arredondar números decimais.

a) Math.round()

Retorna o **número inteiro mais próximo** do número decimal indicado.

Exemplo:

```
System.out.println( Math.round(3.49) );
System.out.println( Math.round(3.1) );
System.out.println( Math.round(3.501) );
```

Resultado:

```
3
3
4
```

b) Math.floor()

Retorna o **número inteiro abaixo** do decimal indicado

Exemplo:

```
System.out.println( Math.floor(7.99) );
System.out.println( Math.floor(7.01) );
```

Resultado:

```
7
7
```

c) Math.ceil()

Retorna o **próximo número inteiro acima** do decimal apontado.

Exemplo:

```
System.out.println( Math.ceil(7.01) );
System.out.println( Math.ceil(7.99) );
```

Resultado:

```
8
8
```

* Para reforçar o que você acabou de aprender neste capítulo, assista esta videoaula sobre o mesmo assunto no meu canal do YouTube: https://youtu.be/JQzfqZBhuxw

13. Números Aleatórios (*Randômicos*)

Os números aleatórios (*Random*) são utilizados constantemente.

Tente imaginar ocasiões em que eventos aleatórios podem estar acontecendo em aplicações que utilizamos no dia-a-dia:

- Nos ***games***, para realizar variações de eventos e jogadas;
- Na ordenação das postagens que vemos na *timeline* do *Facebook,* por exemplo;
- Na segurança de informação, para a geração de senhas e identificações.

No Java existem algumas maneiras diferentes de se gerar números aleatórios, porém eu gostaria de te mostrar uma bem interessante: através da **Biblioteca Externa Random**.

Esta biblioteca vai nos permitir gerar um número aleatório com muita facilidade.

Vamos aprender como:

1) Para utilizar alguma **Biblioteca Externa no Java**, precisamos primeiro importá-la para nosso código.

Utilize o comando **import** logo no início do seu código.

```
import java.util.Random;
```

2) Declare uma variável do tipo **Random()** e inicialize-a com o comando (construtor) **new**. Mais adiante falaremos mais sobre este construtor.

```
Random meuRandom = new Random();
```

3) Imprima esta variável no Console de Mensagens e veja o que aparece:

```
System.out.println(meuRandom);
```

Exemplo:

```
import java.util.Random;

public class HelloWorld
{
  public static void main(String[] args)
  {
    Random meuRandom = new Random();
    System.out.println(meuRandom);
  }
}
```

Resultado:

```
java.util.Random@7852e922
```

Perceba que o resultado é um objeto com um valor irreconhecível, que não podemos manipular ainda.

É preciso converter o objeto em um Número Inteiro. Para isto utilize o método **nextInt()**

Exemplo:

```
int inteiroRandom = meuRandom.nextInt(5);
```

O resultado no Console será cada vez que executar, um **número aleatório entre 0 e 4**.

O argumento (**5**) indica o intervalo de números que deve ser sorteado. No caso **de 0 a 4** (5 números).

14. Operadores de Comparação

Podemos comparar dois valores e obter ***true*** ou ***false,*** caso a comparação seja satisfeita, ou não.

OPERADOR	COMPARAÇÃO
==	igual
!=	diferente
<	menor
>	maior
<=	menor ou igual
>=	maior ou igual

Observe quais são os **Operadores de Comparação**:

Faça alguns testes no seu compilador e observe os resultados.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    System.out.println(10 == 10);
    System.out.println(true == false);
    System.out.println(5 != 5);
    System.out.println(1 < 10);
    System.out.println(5 > 3);
    System.out.println(10 <= 10);
    System.out.println(5 >= 1);
  }
}
```

Resultado:

```
true
false
false
true
true
true
true
```

15. Operadores Lógicos

As operações lógicas servem para comparar dois valores booleanos e retornar um resultado referente à esta comparação. Existe na programação basicamente 3 Operadores Lógicos:

OPERADOR	NOME	TRADUÇÃO
&&	And	E
||	Or	OU
!	Not	NÃO

a) Operador And (&&)

Vai comparar dois *booleanos* e retornar **true** apenas se o primeiro valor for **true E** o segundo valor também for **true**.

Exemplo:

```
System.out.println(true && true);
System.out.println(true && false);
System.out.println(false && true);
System.out.println(false && false);
```

Resultado:

```
true
false
false
false
```

b) Operador Or (||)

Vai comparar dois *booleanos* e retornar **true** se o primeiro valor for **true OU** o segundo valor for **true**.

Exemplo:

```
System.out.println(true || true);
System.out.println(true || false);
System.out.println(false || true);
System.out.println(false || false);
```

Resultado:

```
true
true
true
false
```

c) Operador Not (!)

Inverte o valor do *booleano*.

Exemplo:

```
System.out.println( ! true );
System.out.println( ! false );
```

Resultado:

```
false
true
```

* Para reforçar o que você acabou de aprender neste capítulo, assista esta videoaula sobre o mesmo assunto no meu canal do YouTube:https://youtu.be/UnhkH8x8jIo

16. Condição If

A **Condição If** é utilizada para controlar o fluxo da aplicação digital. É ela quem diz por qual caminho a aplicação deve seguir sua execução, baseada em uma verificação.

A **Condição If** (Se), faz a execução do programa seguir determinado caminho, dependendo do resultado de um teste lógico.

Se o teste resulta em **true** (verdadeiro) o programa segue executando um determinado trecho de código.

Caso o teste resulta em **false** (falso) o programa ignora este trecho de código, ou até mesmo executa outro determinado trecho.

Veja uma implementação da estrutura de Condição If / Else no Java:

Exemplo:

```
int idade = 16;

if ( idade >= 18 ) {
        System.out.print("Pode Dirigir!");
} else {
        System.out.print("Nao Pode Dirigir!");
}
```

Caso a comparação, que está entre parênteses (idade >= 18), seja satisfeita, o programa vai executar o que está dentro das primeiras chaves **{...}**

Exemplo:

```
if ( idade >= 18 ) {
        System.out.print("Pode Dirigir!");
} else {
        System.out.print("Nao Pode Dirigir!");
}
```

Caso o resultado da comparação seja **false** o programa ignora o que está contido nas primeiras chaves, e executa o que está na segunda, depois do **else**.

Exemplo:

```
if ( idade >= 18 ) {
        System.out.print("Pode Dirigir!");
} else {
        System.out.print("Nao Pode Dirigir!");
}
```

O **else,** vale ressaltar, é opcional! Podemos criar uma condição **if** sem o **else**. Caso a condição seja falsa e o **else** não exista, o programa ignora tudo e continua a execução.

Existe também a estrutura **else if**, onde podemos utilizar uma segunda comparação antes de executar o que existe dentro das chaves do **else.**

Exemplo:

```
If ( idade >= 18 ){
  System.out.print("Pode Dirigir!");
}else if ( idade > 60 ){
  System.out.print("Deve Renovar a Carta!");
}else{
  System.out.print("Nao Pode Dirigir!");
}
```

Baseado neste exemplo, perceba que você pode utilizar quantos **if/else** quiser e comparar diferentes valores e situações.

Observe que o conceito de **Comparações Lógicas** e **Estruturas de Condição** se aplica ao desenvolvimento de qualquer tipo de Aplicação Digital como Dispositivos Eletrônicos, Robótica e VideoGames.

A **Estrutura de Condição** é o que faz parecer que o computador **Pensa** e toma suas **Próprias Decisões**.

17. Escopos ou Limites

A palavra **Escopo** define um **limite** de uma **Ação**, contendo **começo e fim.**

Trata-se de um conceito delimitante, onde podemos trabalhar com **Valores** e **Objetos**.

Entenda que tudo que está dentro das chaves **{...}** faz parte do mesmo contexto.

Observe quantas chaves temos no nosso código:

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
    int idade = 12;
  }
}
```

A estrutura das chaves também representa uma **hierarquia** (conceito de parentesco, ou pai e filho).

Perceba que a função **main** está sendo definida **dentro da classe HelloWorld**.

Observe atentamente o exemplo a seguir:

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
       If ( true )
           {
                int numero = 10;
           }
       System.out.print(numero);
  }
}
```

Neste caso, temos ainda uma estrutura de condição **if** dentro da função **main.**

Se você tentar executar este código, ele vai dar erro. Perceba que a variável está sendo criada dentro da estrutura **if**, que é filha da função **main** e está tentando ser utilizada dentro da **main**.

Diferente do exemplo a seguir, onde a variável foi criada no pai (função **main)**, e acessada no filho (condição **if**).

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args)
  {
       int numero = 10;

       If ( true )
           {
                 System.out.print(numero);
           }
  }
}
```

Uma mudança sutil, mas que faz toda a diferença na montagem da estrutura dos dados do programa.

Lembre-se que sempre que há a presença das chaves **{...},** estamos falando de um contexto dentro do outro.

E não se esqueça de que todo limite deve ter **início** e **fim**. Ou seja, sempre que **abrir a chave**, devemos **fechá-la**!

18. Projeto #1 - Jogo do Par ou Ímpar

Agora, utilizaremos todo conhecimento adquirido até aqui. Criaremos um **Jogo de Par ou Ímpar**. Antes de partir para a programação, é importante que você tenha o problema 100% solucionado na sua cabeça.

Para isso, recomendo que você use o bom e velho lápis e papel para você desenhar esse **Algoritmo** à mão, até visualizar muito bem como ele vai funcionar. Só então comece a resolver as **Linhas de Programação**.

Pense como poderia ser desenvolvido um **Algoritmo** para resolver este problema. Tente dividir todo o processo em etapas. Por exemplo:

- Sortear dois Números Aleatórios entre 0 e 5
- Exibir os Valores das jogadas sorteadas
- Verificar se a soma das jogadas é par ou é ímpar
- Exibir mensagem dizendo quem venceu

Se você absorveu o conteúdo deste Livro passo a passo até aqui, já tem todos os conhecimentos necessários para construir este **Algoritmo**.

Entretanto, lembre-se de que podem existir diferentes maneiras de se escrever um programa ou algoritmo de computador. O importante é chegarmos ao objetivo traçado.

Deixarei o problema aqui como um desafio pra você resolver.

DESAFIO #1

Eu te desafio a utilizar tudo o que você aprendeu até aqui, e criar um Jogo de Par ou Ímpar utilizando a Linguagem Java.

Adiante você tem uma resolução, caso precise de ajuda, mas o mais importante é tentar e errar. Assim que se aprende de verdade!

Lesmo

Tente se esforçar e resolver o desafio antes de olhar a minha resolução.

Uruca

DESAFIO #2

Jogue o jogo que você programou com um amigo, diga à ele que foi você mesmo quem criou: que escreveu este programa, linha por linha de código.

Depois ensine seu amigo a programar. Lembre-se que ensinar também é uma ótima maneira de aprender.

À seguir você pode conferir o meu **Código final comentado** com a minha resolução do problema.

Lembre-se de que existem diversas maneiras diferentes para desenvolver este algoritmo.

O mais importante é que no final você alcance um resultado satisfatório na sua aplicação, utilize o mínimo possível de código, e divirta-se bastante no desenvolvimento.

Boa Diversão e Bons Estudos!

Resolução do Projeto #1 - Jogo do Par ou Ímpar

```
import java.util.Random;

public class ParOuImpar
{
  public static void main(String[] args)
  {
    Random meuRandom = new Random();

    // sorteando dois numeros de 0 a 5
    int jogadaA = meuRandom.nextInt(6); // par
    int jogadaB = meuRandom.nextInt(6); // impar

    /*
      verificando se a soma das jogadas e par
      se o modulo (resto) da divisao por 2
      for igual a zero, o numero e par
    */

    if(((jogadaA+jogadaB) % 2) == 0) {
      System.out.println("Jogador A Venceu!");
    }
    else {
      System.out.println("Jogador B Venceu!");
    }
  }
}
```

19. Estrutura de Condição Switch

Em alguns casos, ao invés de utilizar vários **if, else if**… podemos simplificar o **Código** utilizando a estrutura **switch**. Esta estrutura compara um **Valor** a várias outras possibilidades, e caso a comparação seja verdadeira, o **Comando** determinado é executado.

Exemplo:

```
String nome = "Felpudo";

switch(nome) {
    case "Tito" : System.out.print("Oi Tito!"); break;
    case "Felpudo" : System.out.print("Oi Felpudo!"); break;
    case "Fofura" : System.out.print("Oi Fofura!"); break;
    default: System.out.print("Nao conheco voce!"); break;
}
```

Perceba no **Código** do exemplo, os **Comandos** destacados em negrito fazem parte da estrutura do **switch**.

Entenda cada um dos operadores da estrutura **Switch**:

- **switch()** seleciona a **Variável** a ser comparada.
- **case** compara com o valor do próximo argumento.
- **:** Indica o início do comando a ser executado.
- **default** será executado, caso nenhuma das comparações anteriores tenha sido satisfeita.
- **break** - Indica o fim da ação executada. Faz com que a execução pule para fora do escopo na execução do código.

Adiante veremos que o **break** pode ser utilizado também em outras estruturas na programação, para cessar a execução do escopo atual.

* Para reforçar o que você acabou de aprender neste capítulo, assista esta videoaula sobre o mesmo assunto no meu canal do YouTube:https://youtu.be/cp7hcpwPWJk

20. Projeto #2 - Jo Ken Pô

Já sabemos o que é necessário para desenvolver este jogo do **Pedra, Papel ou Tesoura**. Perceba que a mecânica, a lógica do jogo, é bem similar ao Jogo do Par ou Ímpar. Desafio você a criar sozinho este jogo.

Tenho certeza que neste desenvolvimento muitos conceitos e idéias serão formadas no seu raciocínio. Isto é o mais importante.

Lembrando sempre que existem diversas maneiras diferentes de se resolver um problema como este.

Uruca

DESAFIO #1

Crie já o seu Algoritmo em *Java* do Jogo Jokenpô! Uma dica: as estruturas switch ajudam a diminuir o uso da estrutura if/else.

Mãos à obra! Ao final compare seu algoritmo com a minha solução que está adiante.

DESAFIO #2

Desenvolva novamente o algoritmo do Jokenpô, porém desta vez utilizando uma lógica totalmente diferente do primeiro algoritmo.

Compartilhe sua solução na sua rede social e marque o professor Tito Petri.

Lesmo

Resolução do Projeto #2 - Jo Ken Pô

```
import java.util.Random;

public class JoKemPo {
  public static void main(String[] args)
  {

    /*  0. PEDRA
        1. PAPEL
        2. TESOURA  */

    Random meuRandom = new Random();
    int jogadaA = meuRandom.nextInt(3);
    int jogadaB = meuRandom.nextInt(3);

    System.out.println(jogadaA);
    System.out.println(jogadaB);

    if ( jogadaA == jogadaB ) {
      System.out.println("Empate!");
    } else if ( ( jogadaA == 0 && jogadaB == 2 ) ||
             ( jogadaA == 1 && jogadaB == 0 ) ||
             ( jogadaA == 2 && jogadaB == 1 ) ) {
      System.out.println("Jogador A Ganhou!");
    } else {
      System.out.println("Jogador B Ganhou!");
    }
  }
}
```

21. Repetição for

A estrutura de repetição **for** serve para executar um determinado comando várias vezes seguidas.

Podemos determinar o número de vezes que uma instrução será executada.

Execute a estrutura **for** abaixo, e observe o resultado no console:

Exemplo:

```
for ( int i = 0 ; i <= 10 ; i++ ){
      System.out.println("Valor do Contador: " + i);
   }
```

Resultado:

```
Valor do Contador: 0
Valor do Contador: 1
Valor do Contador: 2
Valor do Contador: 3
Valor do Contador: 4
Valor do Contador: 5
Valor do Contador: 6
Valor do Contador: 7
Valor do Contador: 8
Valor do Contador: 9
Valor do Contador: 10
```

Perceba que o comando **print** foi executado 11 vezes, e durante a execução, a variável numérica chamada de **i** foi incrementada (aumentou de valor) a cada vez que o comando foi executado.

Agora vamos entender o que aconteceu. Observe como foi criada a estrutura do **for**:

Exemplo:

```
for ( int i = 0; i <= 10; i++ ) {
    System.out.println("Valor do Contador: " + i);
  }
```

Entre os parênteses você precisa declarar **três argumentos** que estão separados pelo ponto e vírgula (**;).**

É muito importante entender cada um destes argumentos para utilizar o **for**.

O **primeiro argumento (int i = 0)** cria uma variável do tipo inteira e a chama de **i**. Inicializamos este número por zero, e ele serve como um **contador** para nos indicar quantas vezes a estrutura já foi executada.

Utilizamos a letra **i** por questões de boas práticas de programação. Este **contador** poderia ser declarado por qualquer nome.

O **segundo argumento (i <= 10)** é uma **condição**. Perceba que existe uma **comparação** que deve ser satisfeita (verdadeira). Enquanto esta condição for verdadeira, o que está dentro do **for** será executado.

O **terceiro argumento (i++)** é o **incremento**. Cada vez que a estrutura é executada, a variável **i** vai receber +1 no seu valor.

Ou seja, durante a primeira execução do laço, a variável **i** terá o valor **0**. Durante a segunda, o valor **1**, durante a terceira o valor **2**, e assim sucessivamente.

Podemos também trocar o **i++** por **i--** e fazer a variável **decrementar** (diminuir) seu valor conforme a execução.

Utilizando **i+=2** ou **i+=3** podemos incrementar de "**2 em 2"** ou "**3 em 3"** unidades, e assim sucessivamente.

Veja a seguir mais algumas implementações da estrutura (ou laço) de repetição **for**:

a) Decrementando o Contador:

Exemplo:

```
for ( int i = 10 ; i >= 0 ; i -- ) {
    System.out.println("Valor do Contador: " + i);
}
```

Resultado:

```
Valor do Contador: 10
Valor do Contador: 9
Valor do Contador: 8
Valor do Contador: 7
Valor do Contador: 6
Valor do Contador: 5
Valor do Contador: 4
Valor do Contador: 3
Valor do Contador: 2
Valor do Contador: 1
Valor do Contador: 0
```

b) Incrementando o Contador de 2 em 2

Exemplo:

```
for ( int i = 0; i <= 10; i += 2 ){
    System.out.println("Valor do Contador: " + i);
  }
```

Resultado:

```
Valor do Contador: 0
Valor do Contador: 2
Valor do Contador: 4
Valor do Contador: 6
Valor do Contador: 8
Valor do Contador: 10
```

DESAFIO #1

Crie um algoritmo que imprima no console todos os números pares de 0 a 50.

Lesmo

DESAFIO #2

Bugado

Agora crie um algoritmo que imprima no console todos os números divisíveis por 5 no intervalo de 0 a 100.

Compartilhe esta solução com seus amigos na sua rede social e marque o professor Tito Petri.

22. Repetição while

A estrutura **while** é bem parecida com o **for**, porém a sua implementação é um pouco diferente. Execute o seguinte **Código** abaixo e observe o resultado obtido.

Exemplo:

```
int contador = 0;

while ( contador < 5 ) {
  System.out.println(contador);
  contador ++;
}
```

Resultado:

```
0
1
2
3
4
```

Lembre-se que no **for,** declaramos **três argumentos** diretamente na construção: **Criação do Contador**, **Condição** e **Incremento**.

Exemplo:

```
for( int i = 0; i < 5; i ++ )
// contador // condição // incremento
```

No **while**, também precisamos de três **Argumentos**, porém, eles são criados e manipulados separadamente.

Exemplo:

```
int contador = 0;     // contador

while(contador < 5)     // condição

contador ++;      // incremento
```

Preste muita atenção ao utilizar a estrutura **while**. Se você esquecer de modificar o **Valor** do **Contador**, a fim de quebrar a condição do **while**, isso fará com que a execução do programa **nunca saia do *loop***. E a isto chamamos de ***loop* infinito**.

Existem poucas diferenças entre se utilizar uma estrutura **for** e **while**.

A estrutura **for** sempre vai exigir que o **Contador** seja uma **nova Variável.**

Já a estrutura **while** pode utilizar uma **Variável já existente** código ou projeto.

O **while** também nos dá mais flexibilidade para manipular o **Contador** em situações mais específicas.

* Para reforçar o que você acabou de aprender neste capítulo, assista esta videoaula sobre o mesmo assunto no meu canal do YouTube:https://youtu.be/TK80dIrasBM

23. Arrays, Listas ou Vetores

São estruturas de dados ordenadas que armazenam vários objetos (variáveis) do mesmo tipo.

Quando você ouvir os termos *Lista*, *Vetor*, *Array*, *Matriz*, perceba que se tratam basicamente do mesmo conceito.

Ao invés de trabalhar com um único valor ou variável, podemos utilizar uma estrutura ordenada onde vários objetos ficam arrumados e são acessados por coordenadas.

Veja o exemplo a seguir:

Ao invés de armazenar um único valor (palavra) dentro de um String, podemos criar uma lista, e armazenar diversas palavras, separando por índices, ou posições, dentro desta lista.

Observe o código à seguir. Ele exemplifica como podemos criar uma lista de palavras (Strings).

Exemplo:

```
String[ ] nomes = {"Felpudo", "Fofura", "Bugado", "Lesmo", "Uruca"};
```

Perceba que ao declarar a variável, utilizamos o **tipo** (String) seguido por **colchetes [ ]**.

O **colchetes** define que a variável criada será um **Array de Strings**.

Depois definimos o nome da variável (*array)* como **lista** ou qualquer outro nome.

Para inicializar a **lista** com algum valor, precisamos utilizar um *array* de objetos.

Este array é definido por **chaves {...}** e irá conter vários objetos do tipo String, separados por uma vírgula entre eles.

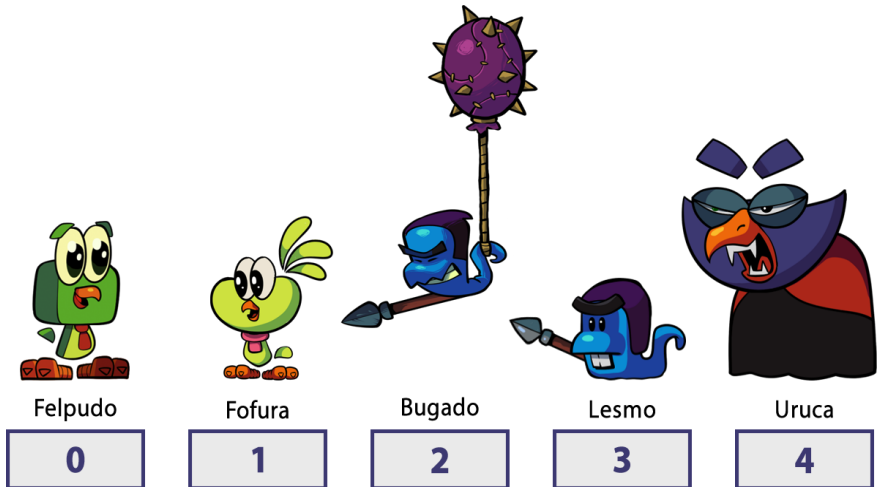
Veja a seguir, a declaração de arrays de outros tipos como **boolean**, **int,** e **float**.

Exemplo:

```
boolean[ ] estados = {true, true, true, false};
int[ ] numeros = {10, 20, 30, 100, 99};
float[ ] decimais = {10.0f, 2.0f, 1.5f, 100.0f };
```

Então lembre-se que para definir uma variável como uma lista, basta adicionar um **[ ]** após o tipo na hora da declaração.

Já armazenamos uma listagem de dados dentro destas variáveis, agora precisamos **acessar** este dado.

Para fazer a **leitura do dado** de uma lista, devemos utilizar o índice da lista.

O índice é a posição de cada objeto dentro do *array*. Lembre-se de que a primeira posição é a casa 0.

Exemplo:

```
String[ ] nomes = {"Felpudo", "Fofura", "Bugado", "Lesmo",
"Uruca"};
```

"Felpudo" - é ocupa o **índice 0** da lista
"Fofura" - é ocupa o **índice 1** da lista
"Lesmo" - é ocupa o **índice 2** da lista
e assim sucessivamente…

Para acessar um índice da lista **nomes** utilizamos o nome da variável e o número do índice entre **colchetes**.

nomes[0] - vai nos trazer o valor contido no primeiro índice (ou posição) da lista.

Exemplo:

```
String[ ] nomes = {"Felpudo", "Fofura", "Lesmo", "Bugado", "Uruca"};

System.out.println( nomes[0] );
System.out.println( nomes[1] );
System.out.println( nomes[4] );
```

Resultado:

```
Felpudo
Fofura
Uruca
```

Perceba que a lista tem **5 itens**, porém, o índice da **última casa é 4**, justamente porque o **índice da primeira casa** é 0.

24. Array Multidimensional ou Matriz

Além das listas, lembre-se também que existem as Matrizes de Dados, ou Arrays Multidimensionais.

As Matrizes utilizam o conceito de organização em grade. Seria como acessar um objeto referenciando o índice da **linha** e da **coluna.** Veja o exemplo:

Exemplo:

```
String [ ][ ] matrizNomes =
{
   {"Felpudo", "Fofura", "Bugado"},
   {"Sao Paulo", "Patopolis", "Salvador"},
   {"ABCD", "EFGH", "IJKL"}
};
```

Perceba que agora a **Matriz** foi declarada utilizando-se **2 colchetes**. E inicializada com um "*array de arrays*", ou

seja, um *array* com 3 *arrays* dentro, separados por vírgula, sempre dentro das chaves.

Para acessar algum dado nesta **matriz**, você deve especificar as 2 coordenadas dos índices.

Exemplo:

```
System.out.print(matrizNomes[0][0]);
System.out.print(matrizNomes[0][1]);
System.out.print(matrizNomes[1][2]);
```

Resultado:

```
Felpudo
Fofura
Salvador
```

No exemplo a seguir, veja como podemos percorrer toda uma matriz, utilizando dois loops de repetição, um dentro do outro, com um contador para as linhas e outro para as colunas.

Exemplo:

```
for(int i = 0; i < matrizNomes.length; i++)
{
    for(int j = 0; j < matrizNomes[ i ].length; j++)
        {
            System.out.println(matrizNomes[ i ][ j ] );
        }
}
```

Resultado:

```
Felpudo
Fofura
Bugado
Sao Paulo
Patopolis
Salvador
ABCD
EFGH
IJKL
```

Chamamos este tipo de procedimento de ***nested loop*** ou **"loops aninhados"**.

Significa que a cada execução do **for**, haverá um outro **for** acontecendo.

Usamos este tipo de procedimento para varrer as **linhas** e **colunas** de uma matriz.

25. Métodos do Array

Conheça algumas das propriedades e métodos que os objetos do tipo *array* possuem.

a) Atributo *length*

Retorna o número de ítens da lista, ou seja, o **tamanho** do *array.*

Exemplo:

```
String[] nomes = {"Tito", "Felpudo", "Fofura"};
System.out.println(nomes.length);
```

Resultado:

```
3
```

Sendo assim, podemos facilmente acessar sempre o **último item** da lista, utilizando o comando:

Exemplo:

```
String[] nomes = {"Tito", "Felpudo", "Fofura"};
System.out.println( nomes [ nomes.length - 1 ] );
```

Resultado:

```
Fofura
```

Independentemente do número de itens existentes, se obtivermos o **tamanho da Lista**, podemos subtrair **-1** e saber qual o **Índice do último item da Lista**.

b) Método *Arrays.toString()*

Veja o método de criação de um array, de uma outra maneira. Perceba que, como ele foi inicializado mais adiante no código, utilizamos o construtor **new** indicando o **tamanho do array**, no caso **três posições**.

Exemplo:

```
String[] nomes;
nomes = new String[3];
nomes[0] = "Tito";
nomes[1] = "Felpudo";
nomes[2] = "Fofura";
System.out.println(nomes);
```

Resultado:

```
[Ljava.lang.String;@6d06d69c
```

Se imprimir o objeto *array* no console, vai perceber que este tipo de objeto tem um valor irreconhecível. Porém, podemos converter este valor em um String utilizando o método **Arrays.toString()**.

Este método exige a **importação da biblioteca** correspondente.

Exemplo:

```
import java.util.Arrays;
...
Arrays.toString(nomes)
```

Resultado:

```
[Tito, Felpudo, Fofura]
```

26. Métodos de Strings

Agora que já conhecemos um pouco sobre *arrays* e os seus métodos, perceba que os *Strings* são objetos do tipo lista.

Podemos considerar os *Strings* como um um **array de caracteres**.

Perceba que também possuem métodos semelhantes aos objetos do tipo *array*:

a) Método length()

Retorna o número de caracteres do **String**, assim como nos outros *arrays*.

Exemplo:

```
String mensagem = "Oi eu sou o Tito Petri!";
System.out.println(mensagem.length( ) );
```

Resultado:

```
23
```

b) Método indexOf()

O método **.indexOf** retrona o **índice** de onde inicia o **String** especificado.

Caso não encontre nenhuma ocorrência, retorna -1.

Exemplo:

```
String mensagem = "Oi eu sou o Tito Petri!";
System.out.println(mensagem.indexOf( "Tito" ) );
```

Resultado:

```
12
```

c) Método charAt()

O método **.charAt()** retorna o **Caractere** do **Índice** especificado.

Exemplo:

```
String msg = "Oi eu sou o Tito Petri!";

System.out.println(msg.charAt(0));
System.out.println(msg.charAt(12));

// acessando o ultimo caractere do String
System.out.println(msg.charAt(mensagem.length() - 1));
```

Resultado:

```
O
T
!
```

d) Método split()

O método **.split** converte o *String* em um *array* de *Strings*, separando-os por um caractere específico.

Para utilizar, primeiro importe a pasta inteira da **biblioteca util**. Utilize o operador * para isso.

Exemplo:

```
import java.util.*;

String mensagem = "Oi eu sou o Tito Petri!";

String[] listaPalavras = mensagem.split("\\s");

System.out.println(Arrays.toString(listaPalavras));
```

Resultado:

```
[Oi, eu, sou, o, Tito, Petri!]
[O,  eu sou o T, to Petr, !]
```

O operador **"\\s"** faz o método **split** quebrar cada palavra do String de acordo com os caracteres do **espaço**.

Observe agora utilizando o caractere **"i"** para separar a frase:

Exemplo:

```
String[] listaPalavras = mensagem.split("i");
```

Resultado:

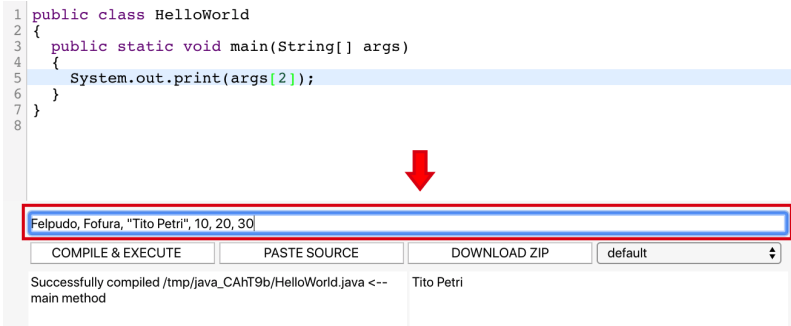
```
[O,  eu sou o T, to Petr, !]
```

27. Argumentos da Linha de Comando

Agora que já conhecemos um pouco sobre as **Listas,** você pode estar se perguntando o que é o **String [ ] args** que observamos na Função Principal (**main**) do nosso **Código**.

```
1  public class HelloWorld
2  {
3     public static void main(String[] args)
4     {
5        System.out.print(args);
6     }
7  }
```

Trata-se de uma linha de parâmetro que podemos passar para executar o nosso programa.

Para isso, use o campo **command line arguments**, logo acima do botão **Compile & Execute**.

Você pode inserir alguns valores na linha de comando de argumentos separando-os por **espaços** ou **vírgulas**.

E pode ler estes valores dentro da função **main** do seu programa. Perceba que eles são passados para a função por um **array de Strings** chamado **args**.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[ ] args)
  {
    System.out.print( args[2] );
  }
}
```

28. Percorrendo uma Lista com for

Agora vamos utilizar um método para percorrer um lista e acessar cada um dos itens armazenados nesta lista.

Para realizar este procedimento, devemos utilizar a **estrutura de repetição for** que aprendemos anteriormente.

Veja no exemplo abaixo, como utilizar uma estrutura **for** para percorrer a lista dos argumentos passados pelo console para o *array* **args**.

Exemplo:

```
public class HelloWorld
{
  public static void main( String[ ] args)
    {
    for(int i = 0; i < args.length; i++)
      {
        System.out.println( "Valor do Indice " + i + " : " + args[ i ]);
      }
    }
}
```

Resultado:

```
Valor do Indice 0 : Felpudo
Valor do Indice 1 : Fofura
Valor do Indice 2 : Tito Petri
```

```
Valor do Indice 3 : 10
Valor do Indice 4 : 20
Valor do Indice 5 : 30
```

O **for** irá percorrer a lista do índice (**i**) zero até o último item do *array* ***args*** (**args.length**) imprimindo o valor de cada item (**args[i]**) no console.

Esta é uma forma bem "artesanal" de fazer um for em uma lista.

Aproveite bem as idéias e conceitos aprendidos aqui, mas saiba que existe no Java um recurso mais simples para varrer uma lista com um **for**.

Exemplo:

```
for( String meuItem : args ) {
    System.out.println( meuItem );
}
```

Podemos percorrer um array com outro tipo de estrutura **for**.

Entre parênteses os argumentos declarados são agora diferentes.

O **primeiro argumento** é o tipo da variável que vai receber cada item do *array* (*String **meuItem***).

A partir daqui, durante a execução do **for**, cada item será lido e guardado em uma variável chamada **meuItem** por exemplo.

O **segundo argumento** depois dos dois pontos (**:**) é o *array* que você deseja percorrer (**args).**

Pare e perceba quantos conceitos envolvidos apenas em uma pequena instrução.

Reflita sobre o tanto de conceitos e idéias envolvidos você precisou entender para interpretar este código.

Muitas destas idéias são regras e práticas universais que você vai usar pra sempre, mesmo se um dia estiver utilizando outra linguagem de programação. Veja que grande habilidade você adquiriu!

29. Projeto #3 - Sorteio de Nomes

Já temos todo o conhecimento necessário para desenvolver a nossa próxima aplicação:

Um programa para **Sortear Nomes** à partir de uma lista!

Pense e desenvolva um algoritmo que seja capaz de:

a) Obter uma **Lista** de nomes (inserido na **linha de argumentos** do console).

b) Sortear um dos nomes da **Lista**.

DESAFIO #1

Desafio você a desenvolver sozinho o Algoritmo do Sorteio de Nomes antes de olhar a minha resolução.

Tentar e errar, esta é a melhor maneira de fixar o conhecimento!

Lesmo

Lembre-se sempre de que existem várias maneiras de desenvolvimento, tente encontrar a sua!

Dicas Importantes!

a) Não se esqueça de **separar os itens por vírgulas** na hora de inserí-los no *command line*.

b) Veja abaixo um exemplo de como você pode inserir os itens na linha de comandos ou argumentos.

Exemplo:

```
Felpudo, Fofura Bugado, "Tito Petri", Lesmo
```

c) Repare que a **vírgula** ou **espaço** definem a **separação dos itens** no *array* de argumentos.

d) E os **nomes compostos,** devem ser inseridos entre **aspas duplas**.

Bugado

DESAFIO #2

Crie um sorteio entre seus amigos no Whatsapp.

Para executar o sorteio, utilize o algoritmo em Java que você mesmo criou.

Mostre o seu código para que os participantes vejam que a aplicação é válida.

Veja a seguir a minha resolução para este algoritmo.

Resolução do Projeto #3 - Sorteio de Nomes

```
import java.util.Random;

public class SorteioDeNomes
{
  public static void main( String[ ] args)
  {
    Random rand = new Random();
    int randomNum = rand.nextInt( args.length );

    System.out.print ( "Nome Sorteado: " + randomNum + " "
                       + args[ randomNum ] );
  }
}
```

30. ArrayList

Outro ponto a ser destacado, é que os simples *arrays* convencionais que utilizamos até aqui podem ser também um pouco limitados.

Só servem para alguns **poucos tipos de dados** e seu **tamanho** é definido apenas na hora da criação do *array,* tornando-se impossível modificar depois.

Observe o exemplo a seguir, duas maneiras diferentes de declarar um simples *array*:

Exemplo:

```
String[ ] nomes;
nomes = new String[3];

String[ ] minhaLista = {"Tito", "Felpudo", "Fofura", "Lesmo",
"Bugado"};
```

Nas duas declarações, perceba que já foi estipulado o número de casas que o Array possuirá.

Não podemos mais adicionar ou remover nenhum índice neste *array*.

Aqui entra um tipo de *array* mais dinâmico e moderno, os ***ArrayLists***.

O **ArrayList** permite que trabalhemos com *arrays* de uma forma bem mais simples e dinâmica.

Por exemplo, você pode **adicionar ou remover um item** específico do *array* e trabalhar com listas de **qualquer tipo de objeto ou variável**.

Mais adiante estaremos criando **objetos de dados mais complexos**, e seremos obrigados a utilizar *ArrayLists* na maior parte do tempo.

Para utilizar o *ArrayList*, importe suas bibliotecas:

Exemplo:

```
import java.util.List;
import java.util.ArrayList;
```

Declare um objeto do tipo ArrayList.

Exemplo:

```
ArrayList<String> novaLista = new ArrayList<String>();
```

Veja outra maneira de criar um *ArrayList* de *String,* primeiro criando o objeto, e inicializando-o logo em seguida no código:

Exemplo:

```
ArrayList<String> novaLista;
novaLista = new ArrayList<String>();
```

Observe que nos dois casos, iniciamos o *Arraylist* pelo método construtor ***new.***

Agora que o *ArrayList* foi criado e inicializado, podemos adicionar os valores na lista.

Este *ArrayList* só vai aceitar que se adicione objetos do tipo *String*.

O tipo do ArrayList foi definido no momento da sua criação (com o operador **<String>**).

Exemplo:

```
import java.util.*;

public class HelloWorld
{
  public static void main(String[] args)
    {
    ArrayList<String>novaLista = new ArrayList<String>();

    novaLista.add("Homer");
    novaLista.add("Chaves");
    novaLista.add("Leonardo");

    System.out.println(novaLista);
    }
}
```

a) Array as List

Também podemos Inicializar o ArrayList de forma ainda mais resumida utilizando o método Arrays.asList():

Exemplo:

```
ArrayList<String> listaNomes = new
ArrayList<>(Arrays.asList("Felpudo","Fofura","Lesmo","Bugado"));
```

Agora podemos declarar, inicializar, e adicionar os valores ao ArrayList em uma única linha de comando.

b) Adicionando e Removendo Itens

No início do capítulo vimos que o comando add adicionava um novo item à Lista. Agora, vamos entender melhor este conceito. Acrescente o seguinte Código abaixo, e volte a imprimir o resultado.

Exemplo:

```
listaNomes.add(1,"Petri");
```

Veja que um novo item é adicionado à Lista, na posição determinada pelo valor da add. e Removendo.

Da mesma forma, para Removermos um item de uma Lista, é preciso declarar o Comando remove(), indicando o Índice (a posição) do item que se deseja eliminar.

Exemplo:

```
listaNomes.remove(2);
```

Ainda é possível pedir para que o programa exiba um determinado item de uma posição, declarando seu Índice usando o Comando get()

Exemplo:

```
System.out.println(listaNomes.get(1));
```

Ou então, saber quantos itens há na Lista, com o Comando size().

Exemplo:

```
System.out.println(listaNomes.size());
```

31. Projeto #4 - Busca por Nome e ID

Neste momento, um belo desafio pra você juntar tudo o que vimos até aqui seria montar uma aplicação que busque um nome em uma Lista, associando este nome ao seu respectivo número de registro de matrícula (ID).

Nossa Lista contém os nomes: Tito, Felpudo, Fofura, Lesmo e Bugado. Os IDs de cada um são respectivamente: 1, 12, 33, 13, 27.

Utilize ArrayLists para armazenar os Dados, e o Comando "break" para sair da busca ao encontrar o item.

Para realizar a busca de um nome qualquer, você deverá digitá-lo no campo **command line arguments**, antes de executar a aplicação. Ao final, a aplicação deve dizer se foi ou não encontrado, imprimir o nome pesquisado, e qual o número de identificação (ID) que atribuímos a este nome.

Tente desenvolver esta aplicação, e depois confira a minha resolução.

Solução para o Projeto #4 - Busca por Nome e ID

```
import java.util.*;

public class BuscarNomeID
{
    public static void main(String[] args)
    {

ArrayList<String> listaNomes = new
ArrayList<>(Arrays.asList("Tito","Felpudo","Fofura","Lesmo","Bugado"));

ArrayList<Integer> listaIDs = new
ArrayList<>(Arrays.asList(1,12,33,13,27));

boolean encontrado = false;

for( int i = 0; i < listaNomes.size(); i++ )
      {
           if(listaNomes.get(i).equals(args[0]))
           {
                 encontrado = true;
                 System.out.println("Nome Encontrado: "
                 + listaNomes.get(i) + "\nID: " + listaIDs.get(i));
                 break;
           }

          if (!encontrado)
          {
               System.out.println("Nome NÃO Encontrado: " + args[0]);
          }
    }
}
```

Dicas Importantes:

Utilize **2 *ArrayLists***, um para armazenar os **nomes** e o outro os **IDs**.

A variável booleana chamada de **encontrado** servirá para nos indicar que o item foi achado na lista.

Ela permanecerá falsa até que o nome seja encontrado na lista.

Caso nenhum nome seja encontrado, exibir uma **mensagem de aviso**.

32. Métodos e Procedimentos

Métodos são blocos de comandos que usamos para resumir e encapsular uma ação.

Vamos abstrair e imaginar a seguinte situação:
Alguém entrando em um carro.

Esta ação envolveria algumas instruções:

Podemos resumir todas estas ações chamando apenas um comando: **entrar no carro**. Isto simplificaria muito as coisas no caso da construção de um algoritmo. Cada vez que a **função** ou **método** fosse chamado, todas as instruções relativas à este método seriam executadas.

Veja a seguir o exemplo de como podemos **criar ou registrar uma nova função** em nosso código.

Exemplo:

```
public static void Falar( ) {
    System.out.println("Oi eu sou o Felpudo!");
    System.out.println("Eu vou te ajudar a Aprender Java!");
}
```

Basta declarar o método como posto acima, dentro do escopo da classe em que você está trabalhando. Até aqui, apenas **criamos o método**. Precisamos agora **executá-lo**. Para isso basta usar o nome do método:

Exemplo:

```
Falar();
```

Veja a seguir como ficou a **declaração** e a **utilização** do método que acabamos de criar

Exemplo:

```
public class HelloWorld {
  public static void main(String[] args) {
     Falar();
  }
  public static void Falar() {
    System.out.println("Oi eu sou o Felpudo!");
    System.out.println("Eu vou te ajudar a Aprender Java!");
  }
}
```

Agora temos o método criado e cada vez que usarmos o comando **Falar( )**, o bloco inteiro do método será executado.

Preste atenção em alguns detalhes: existem dois tipos de métodos. **Com** e **Sem Retorno** de valor.

Podemos também executar métodos passando **valores**, ou **parâmetros** como **argumentos.**

Uma coisa é você instruir o Joãozinho a ir até a padaria. Outra coisa, é você instruí-lo a ir na padaria, e comprar **5 pães** e **1 leite.**

Podemos entender que os **pães** e **leite** são os argumentos do método, os parâmetros que foram dados para que o método fosse executado.

Veja a seguir como seria a utilização de um método com **passagem de argumentos**.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args) {
        Falar("Oi eu sou o Tito Petri!");
  }
```

```
  public static void Falar( String mensagem) {
    System.out.println(mensagem);
  }
}
```

Agora declaramos qual será o argumento necessário para executar o método (**String mensagem**).

E, para executar o método Falar(), precisaremos sempre passar um **argumento** ou **parâmetro** do tipo String entre os parênteses.

Observe agora, um segundo exemplo em que iremos passar **2 argumentos** para o método a ser executado.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args) {
      Somar(10,35);
  }

  public static void Somar(int a, int b) {
    System.out.print(a + b);
  }
}
```

Perceba que na hora da declaração (criação) do método, definimos o **tipo** da **variável** e o **nome**, separando cada **argumento** por vírgula.

33. Função ou Método com Retorno

Até então, criamos métodos que apenas são executados e acabam ali.

Existem também o que chamamos de **função**, que é basicamente o mesmo que um método. Porém, a **função** nos devolve algum valor como resultado final da sua execução.

Até então, sempre declaramos nossos métodos usando a palavra **void**, isto significa que o **retorno** da função é **vazio**, ou seja, ela **não tem retorno.**

Observe agora como é uma função, ou método com retorno de valor.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args) {
      int resultado = Multiplicar(12,5);
  }

  public static int Multiplicar(int x, int y) {
    return ( x * y );
  }
}
```

Perceba que agora a função **não foi declarada como void** e sim como **int**, que é o tipo de valor que esta função deve retornar através do comando **return**.

Se você executar este código, verá que nada vai acontecer, a função **Multiplicar()** agora apenas executa a multiplicação, e retorna o resultado, que vai ser guardado na variável **resultado**.

Lembre-se então:

a) Método
É o encapsulamento de uma sequência de ações.
Funções e Procedimentos são considerados Métodos!

b) Função
É um método que necessariamente **retorna um valor**.

c) Procedimento
É o método que **não tem retorno**, no caso, declarado como **void**.

É comum os desenvolvedores não darem atenção à este detalhe, e saírem chamando tudo de **função**. Ponto para você que agora você já sabe exatamente qual é a diferença!

34. Projeto #5 - Validação do CPF

Existe uma norma para se definir a geração de números do CPF. É um **Algoritmo** já bem conhecido entre os programadores e estudantes de Ciência da Computação.

Trata-se de ler os 9 primeiros dígitos e calcular os dois últimos dígitos da sequência total de 11 números. Vamos entender como é feito este cálculo:

Vamos utilizar o CPF 123.456.789. Os dígitos finais para que este CPF seja válido devem ser **0** e **9**.

Vamos entender os passos para este algoritmo funcionar:

1) Primeiro, precisamos multiplicar os valores dos 9 digitos pelos números de 10 a 2 como na grade abaixo:

CPF	1	2	3	4	5	6	7	8	9
x	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*
=	**10**	**18**	**24**	**28**	**30**	**30**	**28**	**24**	**18**

2) Agora some todos os resultados, e obtenha o **Resto da Divisão** por **11**.

Se o número for **menor que 2**, o **Primeiro Dígito** é considerado o número **Zero**!

Se o número **for 2 ou mais**, subtraia o resultado de **11**.

Soma Total	Divisão	Resto da Divisão	Primeiro Dígito
210	11	1	0

Agora repita o processo de soma total, **inserindo o primeiro dígito** encontrado ao final da sequência, e fazendo a **multiplicação de 11 até 2** desta vez.

CPF	1	2	3	4	5	6	7	8	9	0
x	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*
=	**11**	**20**	**27**	**32**	**35**	**36**	**35**	**32**	**27**	**0**

Veja como fica o resultado final da validação deste CPF:

Soma Total	Divisão	Resto da Divisão	Segundo Dígito
255	11	9	9

Agora o que sabemos é que baseado nos 9 primeiros dígitos do CPF. Ex.: 123456789. Podemos calcular qual terá que ser os últimos 2 dígitos para que este CPF seja válido. No caso sabemos que é o **0** e o **9**.

É muito importante que você entenda o funcionamento do algoritmo muito bem antes de começar a sua programação.

Aconselho que você faça antes os cálculos em um **pedaço de papel**.

Escreva os valores no formato de uma tabela como a que temos aqui no Livro, e faça os cálculos para chegar no resultado.

Estes cálculos manuais, chamamos de **Teste de Mesa**. Eles são muito importante para nos assegurarmos do funcionamento do algoritmo antes de partirmos para os códigos e a programação.

Use o seu CPF ou de alguns amigos e faça alguns testes.

Depois confira a seguir a minha resolução para este algoritmo.

Resolução para o Projeto #5 - Validação do CPF

```
import java.util.*;

public class ValidacaoCPF {

  public static void main(String[] args) {
    int[ ] numeros;

    numeros = ConverteNumeros(args);
    CalculaDigitos(numeros);
    ImprimeValorCPF(numeros);

  }

/*
   A funcao converte numeros serve para transformar o array de Strings que
vem
   do command line arguments e transforma-lo em um array de ints
*/

  public static int[ ] ConverteNumeros(String[] numerosCPF)  {

    int[] numerosInt;
    numerosInt = new int[11];

    for(int i=0; i<numerosCPF[0].length();i++) {
      char letraVez = numerosCPF[0].charAt(i);
      String stringVez = String.valueOf(letraVez);
      int digitoNumero = Integer.parseInt(stringVez);
      numerosInt[i] = digitoNumero;
    }
    return numerosInt;
  }

  public static int[] CalculaDigitos(int[] numerosCPF)  {
    int iniciaMultiplicacao = 10;
    int primeiraSoma = 0;
    int primeiroDigito = 0;

    for(int i=0; i<9; i++) {
      primeiraSoma = primeiraSoma + (numerosCPF[i] * iniciaMultiplicacao);

      iniciaMultiplicacao--;
    }
```

```
    int resto = primeiraSoma%11;

    if(resto<2) {
      numerosCPF[9] = 0;
    } else {
      numerosCPF[9] = 11 - resto;
    }
    iniciaMultiplicacao = 11;
    primeiraSoma = 0;
    for(int i=0; i<10; i++) {
      primeiraSoma = primeiraSoma + (numerosCPF[i]*iniciaMultiplicacao);
      iniciaMultiplicacao--;
    }
    primeiroDigito = primeiraSoma%11;
    if(primeiroDigito<2) {
      numerosCPF[10] = 0;
    } else {
      numerosCPF[10] = 11 - primeiroDigito;
    }
    return numerosCPF;
  }

/*
    A funcao imprime valor apenas imprime o cpf no console
    ja no formato de um CPF com os pontos e digitos no final
*/

public static void ImprimeValorCPF(int[] numerosCPF) {
    System.out.println("Digitos Validos: " + numerosCPF[9] + numerosCPF[10] + "\n");

    for(int i = 0; i<numerosCPF.length; i++) {
      if((i%3==0)&&(i>0)&&(i<9)) {
        System.out.print(".");
      }
      if(i==9){
        System.out.print("-");
      }
      System.out.print( numerosCPF[ i ]);
    }
  }
}
```

35. Projeto #6 - Números Primos

Os **Números Primos** são números inteiros **divisíveis** apenas **por 1** ou **por ele mesmo!**

Os números 2, 3, 5, 7, 11, 13 são exemplos de **números primos** pois são divisíveis apenas por **eles mesmos** ou **por 1**.

DESAFIO #1

Eu te desafio a utilizar tudo o que você aprendeu até aqui, e Desenvolver um algoritmo para calcular todos os números primos de zero até um determinado número inserido na linha de comando do compilador.

Lesmo

Para resolver este desafio, você vai ter que varrer uma lista do número zero até o número inserido pelo usuário e dividi-lo por todos os números menores que ele.

Para fazer esta verificação, veja no meu exemplo como eu resolvi utilizando dois loops de repetição um dentro do

outro, assim como fizemos no capítulo *"2. Array multidimensional ou Matriz"* para varrer uma **tabela ou matriz de dados.**

Aproveite e observe a foto e a piada a seguir:

Caso você não entenda a gag ou *meme* ou piada à seguir, sugiro que estude mais os laços de repetição, ou então reformule seu senso de humor.

Agora que você já entendeu bem sobre o **aninhamento de loops** ou ***nested loops*** (conceito de um loop de repetição for dentro do outro).

Observe a seguir a minha resolução para o problema dos números primos:

Resolução para o Projeto #6 - Números Primos

```
import java.util.*;

public class NumerosPrimos
{
  public static void main(String[] args)
  {

    ArrayList<Integer> listaNumeros = new ArrayList<Integer>();

    //convertendo o argumento de String para int
    int numeroMaximo = Integer.parseInt(args[0]);

    System.out.println("Primos de Zero a " + numeroMaximo + " :\n");

     // adicionando os numeros em uma lista
    for(int i = 0; i <= numeroMaximo; i++) {
      listaNumeros.add(i);
    }


     // verificando a lista
    for(int i = 0; i < listaNumeros.size(); i++) {

      boolean primo = true;

      // loop aninhado para executar as divisoes

      for(int j = 2; j < listaNumeros.get(i); j++)
      {
        if(listaNumeros.get(i) % j == 0)
        {
          primo = false;
          break;
        }
      }
```

```
    // caso booleano verdadeiro, imprime numero como primo.

     if(primo == true) {
       System.out.print(listaNumeros.get(i)+ ", ");
     }
   }
 }
}
```

Para executar o código acima, **insira um número na linha de argumento** e compile a aplicação.

O resultado obtido no console será todos os números primos de 0 até o seu número digitado.

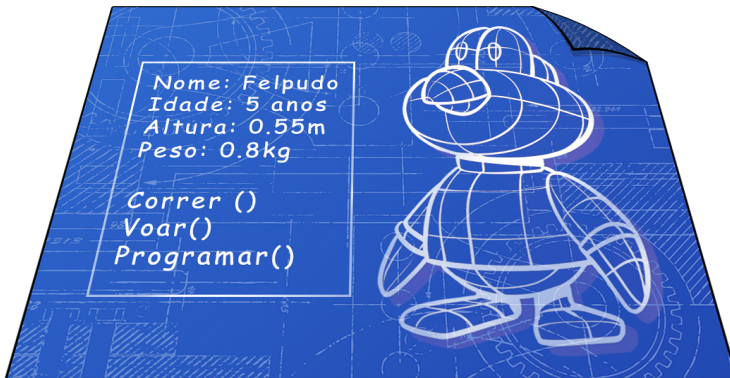
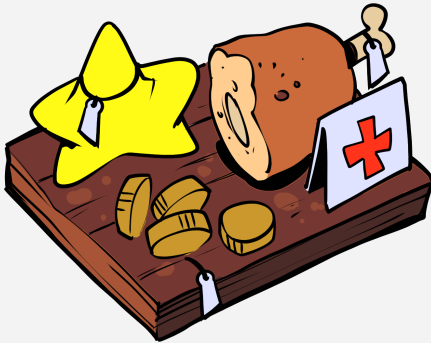
Resultado:

```
Primos de Zero a 100 :

0, 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 83, 89, 97,
```

36. Classes no Java

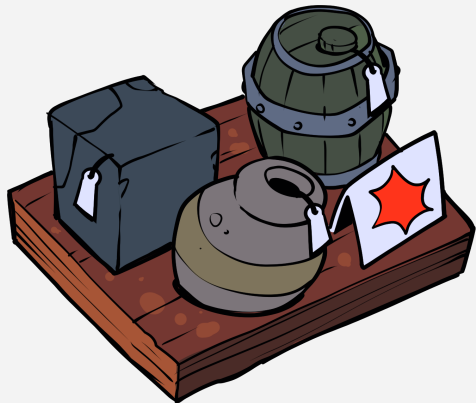
As **classes** são estruturas que servem de modelos para criarmos objetos.

Imagine uma receita, um molde, que reúne todas as características e métodos à respeito da formação de um determinado **objeto.**

A **classe** é praticamente o modelo que reúne os **métodos** e **atributos** refrentes à um determinado **objeto.**

Cada objeto tem suas próprias características (atributos, variáveis) e métodos de funcionamento.

Ao criar ou registrar a classe, você define todos estes métodos e atributos uma única vez.

Depois pode reutilizar estes métodos e atributos pois a cada objeto criado na sua programação, ele já vai vir com todos estas propriedades.

Para "fabricar" um objeto dentro da nossa programação, precisamos primeiro especificar sua **classe**.

Com base nesta classe podemos criar **novos objetos** através do método **new**.

Veja alguns exemplos de onde o método **new** já foi utilizado anteriormente neste Livro:

```
Random meuRandom = new Random();

nomes = new String[3];

ArrayList<String> novaLista = new ArrayList<String>();
```

Ou seja, todos estes objetos são algum tipo de classe, aonde para criar o **objeto** da classe precisamos utilizar o **método construtor new**.

Vamos a seguir criar e utilizar uma nova classe no Java:

Exemplo:

```
public class HelloWorld
{
  public static void main(String[ ] args)
  {
    Animal animal = new Animal();
  }
}

class Animal
{
  String nome;
  int idade;
  float peso;
  String cor;
}
```

Perceba que já existia uma classe no nosso programa, a classe principal chamada **HelloWorld**.

A partir do comando **class** definimos uma nova classe e criamos os parâmetros e atributos dentro das chaves **{...}**

Exemplo:

```
class Animal {

    // Definição dos Atributos:

    String nome;
    int idade;
    float peso;
    String cor;

    // Definição de Método e Função

    void Falar( String nome ){
        System.out.println("Ola eu sou o " + nome);
    }

    void Andar( int distancia){
        System.out.println("Andou " + distancia + " metros.");
    }

    String Correr(){
        return "#partiu!";
    }
}
```

E para instanciar (ou criar) um objeto na sua programação, utilize o método construtor, e declare o

objeto seguindo as regras que utilizaria para criar uma variável qualquer.

Primeiro o **tipo**, depois o nome da variável, e depois sua inicialização (onde teremos que utilizar o construtor **new).**

Veja no exemplo a seguir como criar um novo objeto e manipular seus métodos e atributos.

Exemplo:

```
// criando um novo objeto da classe animal
Animal animal = new Animal();

// inicializando os atributos
animal.nome = "Azul";
animal.idade = 12;
animal.peso = 10.0f;
animal.cor = "Verde";

// executando os metodos
animal.Falar("Felps");
animal.Andar(50);
String msg = animal.Correr();
System.out.print(msg);
```

Perceba que o objeto foi criado e logo em seguida, inserimos os valores nos seus atributos.

Também podemos utilizar o **Método Construtor**, para já criar e parametrizar o objeto logo no comando da criação.

Para isto, na sua classe você deve utilizar o metodo construtor da classe:

Exemplo:

```
public Animal(String nome, int idade, float peso, String cor) {
  this.nome = nome;
  this.idade = idade;
  this.peso = peso;
  this.cor = cor;
}
```

Observe que através deste método, que deve ser um método público e ter o mesmo nome da classe, estamos passando quatro argumentos como parâmetro. Estes argumentos serão utilizados para inicializar os atributos da classe, que são referenciados por **.this.**

Agora na hora de criar o objeto, utilizamos o seguinte método:

Exemplo:

```
  Animal animal = new Animal( "Felpudo", 12,1.5f, "Verde" );
```

Assim seu objeto já está criado e inicializado.

Confira o código inteiro da implementação da classe já com o seu **Método Construtor**.

Exemplo:

```
public class HelloWorld
{
  public static void main(String[] args)
  {

    Animal animal = new Animal("Felpudo",12,1.5f,"Verde");

    System.out.println(animal.nome);

    animal.Falar("Felps");
    animal.Andar(50);
    String msg = animal.Correr();
    System.out.print(msg);
  }
}

class Animal
{
  String nome;
  int idade;
  float peso;
  String cor;

  public Animal(String nome, int idade, float peso, String cor) {
    this.nome = nome;
    this.idade = idade;
    this.peso = peso;
    this.cor = cor;
  }
```

```
  void Falar( String nome ){
    System.out.println("Ola eu sou o " + nome);
  }

  void Andar( int distancia ){
    System.out.println("Andou " + distancia + " metros.");
  }

  String Correr(){
    return "#partiu!";
  }
}
```

37. Hierarquia de Classes

Hierarquia é o conceito de parentesco (pai e filho) que criamos entre as classes.

Uma classe pode herdar todos os parâmetros e atributos de uma classe superior.

Vamos usar de exemplo um ser humano. Ele pode herdar todas as características de um animal qualquer, porém, ele pode também ter suas próprias atribuições e funções específicas.

Observe como podemos extender uma classe tanto aqui quanto no caso anterior:

Exemplo:

```
class Humano extends Animal{

  String profissao;
  String carro;
  int saldo;

  public Humano(String nome, int idade, float peso, String cor) {
    super(nome, idade, peso, cor);
  }

  void Trabalhar(){
    System.out.println("Work!");
  }
}
```

Criamos uma nova classe, e usamos o comando **extend** para estender todos os parâmetros e métodos da classe referida.

No metodo construtor, podemos utilizar o comando **super()** para inicializar os parâmetros da classe superior (no caso a clase animal).

Agora a **Classe Animal** tem seus determinados atributos e métodos, e a **Classe humano** tem todos os atributos e métodos da classe animal, e ainda seus próprios atributos e métodos.

Exemplo:

```
Humano humano = new Humano("Tito Petri",23,1.5f,"Azul");
System.out.println(humano.nome);
```

Entender este conceito de hierarquias facilita muito a estruturação da programação orientada a objeto. E vai tornar nossa programação muito mais simples, organizada e fácil de ser reutilizada.

Perceba que em um game, ou aplicativo, cada janela, componente, objeto, ou personagem pode ser organizado como uma classe para a facilitação do entendimento do código e reaproveitamento das programações.

38. Modificadores de Acesso

São formas de **ocultar** ou **expor** os **Atributos** de **Métodos** das classes.

a) Public

A variável pode ser vista e modificada de todas as outras classes que podem acessar o objeto.

No **modificador público** todos têm acesso.

b) Private

A variável só poderá ser vista e modificada de dentro da **classe** em que foi criada.

A única **Classe** que tem acesso ao **Atributo** é a própria classe que o define.

Ou seja, se uma classe "Pessoa" declara um **Atributo Privado** chamado "nome", somente a classe "Pessoa" terá acesso a ele.

c) Default

Um atributo **default** (identificado pela **ausência de modificadores**) pode ser acessado por todas as classe que estiverem no mesmo pacote em que a classe que possui o atributo se encontra.

d) Protected

Esse **Modificador** é o que mais causa confusão, ele é praticamente igual ao **default**, com a diferença de que se uma classe (mesmo que esteja fora do pacote) estende da classe com o modificador **protected**, esta classe poderá ter acesso a este atributo.

O acesso de um atributo protegido é dado por **pacote** e por **herança**.

39. Modificadores Static e Final

Entenda mais alguns modificadores que alteram as propriedades dos atributos, métodos e classes.

É muito importante conhecer estas normas. Lembre-se que estas regras se aplicam à praticamente qualquer linguagem de programação.

a) Modificador Final - Constantes

Quando definimos uma variável como **final**, ela torna-se uma **Constante**.

Exemplo:

```
final int IDADE = 10;
```

Quer dizer que à partir do momento que você inicializá-la, não conseguirá mais trocar seu valor.

Utilizamos este modificador em parâmetros que nunca serão alterados durante a nossa aplicação.

Por **boas práticas de programação**, geralmente utilizamos **caracteres maiúsculos** para dar nome às constantes.

b) Atributos Estáticos

Quando um atributo é **estático**, significa que seu valor será único para **todos os objetos** criados à partir de uma classe.

Observe que o atributo **idade,** declarado como **static** vai sempre possuir o mesmo valor em qualquer um dos **dois objetos** criados a partir da mesma classe.

Exemplo:

```
public class OlaMundo
{
  public static void main(String[] args)
  {
    Animal animalA = new Animal();
    Animal animalB = new Animal();

    animalA.idade = 10;
    animalB.idade = 20;

    System.out.println(animalA.idade);
    System.out.println(animalB.idade);
  }
}

class Animal{
  String nome;
  static int idade;
  String pais;
  String cor;
}
```

c) Classe Estática

Quando uma classe é definida como **estática,** quer dizer que ela pode ser acessada **sem a necessidade de se criar uma instância** (cópia) do objeto através do construtor.

Exemplo:

```
public class OlaMundo
{
 public static void main(String[] args)
 {
 Animal.MinhaClasseInterna classeInterna = new
Animal.MinhaClasseInterna();
 }
}

class Animal{
  String nome;
  static int idade;
  String pais;
  String cor;

  public static class MinhaClasseInterna {
    public MinhaClasseInterna() {
      System.out.println("Minha Classe Interna Animal!");
    }
  }
}
```

40. Sobrescrita e Sobrecarga de Método

Aqui entram dois conceitos bem importantes na hora de criar os Métodos de uma hierarquia de classes:

a) Sobrescrita de Método - @Override

O Operador **@Override** é uma forma de garantir que você está sobrescrevendo um método, e não criando um novo.

Perceba que no exemplo abaixo, a sub-classe (filho) substituiu o método da super-classe (pai).

Exemplo:

```
public class ClassePai {
    public void imprime() {
        System.out.println("Mensagem Pai!");
    }
}

public class ClasseFilho extends ClassePai {
    @Override
    public void imprime() {
        System.out.println("Mensagem Filho!");
    }
}
```

O que causa um pouco de confusão, é que este operador é opcional, se você removê-lo, perceberá que não vai mudar nada!

Este operador existe, para reforçar o entendimento da estrutura dos métodos nas classes.

Em alguns compiladores, se você usar um operador **@Override** em um método que não existe na classe superior, ele pode dar erro!

b) Sobrecarga de Método - Overload

Quando dizemos que um Método é Sobrecarregado (*Overloaded Method*), significa que este método é utilizado para diferentes funções, de acordo com os parâmetros que receber.

Veja que na classe do exemplo abaixo, definimos três métodos com o mesmo nome. Porém, cada um deles recebe argumentos de diferentes tipos.

De acordo com o argumento passado, o compilador já entende qual o método relacionado.

Exemplo:

```
public class MinhaClasse
{
  public static void main(String[] args)
  {
    Animal animal = new Animal();
```

```
    animal.falar();
    animal.falar("Oi eu sou o Felpudo!");
    animal.falar(12);
  }
}

class Animal {
  void falar(){
    System.out.println("Oi!");
  }
  void falar(String msg){
    System.out.println(msg);
  }
  void falar(int numero){
    System.out.println("Numero Escolhido: " + numero);
  }
}
```

40. Modelo de Dados

A partir de agora, vamos entender um pouco mais sobre a **arquitetura de dados** de um programa.

Vou utilizar como exemplo o cadastro dos dados de uma pessoa. Podemos reunir todos estes dados em um objeto único, estruturando seus atributos no formato de um **modelo de dados**.

Para isso vamos definir uma classe, que vou chamar de **Contato**.

Cada objeto desta classe vai guardar todos os atributos, ou dados que eu preciso para armazenar os **Dados de uma Pessoa**. Basicamente, uma classe com alguns atributos.

A única novidade é que agora vamos usar os *ArrayLists* para armazenar uma lista de **objetos** criados à partir da classe **Contato.**

Exemplo:

```
import java.util.List;
import java.util.ArrayList;

class Contato{

String nome;
  String telefone;
  String email;
  String empresa;

  public Contato(String nome, String telefone, String email, String empresa){
    this.nome = nome;
    this.telefone = telefone;
    this.email = email;
    this.empresa = empresa;
  }
}

public class OlaMundo
{
  public static ArrayList<Contato> minhaLista = new ArrayList<Contato>();
  public static void main(String[] args)
  {
    Contato meuContato = new Contato("Tito
Petri","555-1234","tito.petri@gmail.com","Tito's Coorp.");

    System.out.println(meuContato.nome);
    System.out.println(meuContato.telefone);
    System.out.println(meuContato.email);
    System.out.println(meuContato.empresa);

    minhaLista.add(meuContato);

    System.out.println(meuContato);
    System.out.println(minhaLista.get(0));
  }
}
```

Resultado:

```
Tito Petri
555-1234
tito.petri@gmail.com
Tito's Coorp.
Contato@6d06d69c
Contato@6d06d69c
```

Perceba que qualquer objeto do mundo real possui uma série de características atribuídas à ele (nome, cor, tamanho, peso, idade e outros).

Podemos analisar estas características e modelar o nosso banco de dados à partir delas.

41. Projeto #7 - Cadastro e Busca

Neste próximo algoritmo, nossas programações já começam a ficar maiores, e o nível de abstração aumenta ainda mais.

Serei breve na explicação, para que o próprio leitor pare, se concentre, e tente entender por si mesmo os códigos a seguir que já foram explicados isoladamente ao longo de todo este material.

No programa a seguir, basicamente vamos criar um *ArrayList* de objetos da classe **Contato** e inicializá-lo com alguns contatos.

Observe que também existe a implementação de alguns métodos importantes para trabalhar no Banco de Dados.

- Adicionar Contato
- Remover Contato por Nome
- Buscar Contato
- Imprimir Lista Completa

Exemplo:

```
import java.util.List;
import java.util.ArrayList;

class Contato
{

  String nome;
  String telefone;
  String email;
  String empresa;

  public Contato(String nome, String telefone, String email, String empresa)
  {
    this.nome = nome;
    this.telefone = telefone;
    this.email = email;
    this.empresa = empresa;
  }
}

public class OlaMundo
{
  public static ArrayList<Contato> minhaLista = new ArrayList<Contato>();

  public static void main(String[] args)
  {
```

```
    InserirContato("Felpudo","1234","contato@mail.com","Escola
VideoGame");
    InserirContato("Fofura","1234","contato@mail.com","Escola
VideoGame");
    InserirContato("Lesmo","999-8888","lesmo@mail.com","ACME Inc.");
    InserirContato("Bugado","555-6666","bugado@mail.com","ACME Inc.");
    InserirContato("Uruca","555-6666","uruca@mail.com","ACME Inc.");
    InserirContato("Tito Petri","555-1234","tito.petri@gmail.com","Tito
Coorp.");

    ImprimirLista();

    Contato meuContato = BuscarContatoPorNome(args[0]);

    ImprimirContato(meuContato);
    ApagarContatoPorNome("Fofura");
    ImprimirLista();
  }

  public static Contato BuscarContatoPorNome(String nome)
  {
    int contador=0;
    Contato novoContato = new Contato("n/a","n/a","n/a","n/a");
    for (int i = 0; i < minhaLista.size(); i++)
    {
      String nomeVez = minhaLista.get(i).nome;
      if(nomeVez.equals(nome))
      {
        contador=i;
        System.out.println("NOME ENCONTRADO!");
        novoContato = minhaLista.get(contador);
        break;
      }
      else if(i == minhaLista.size() - 1)
      {
        System.out.println("NOME NAO ENCONTRADO!");
      }
    }
    return novoContato;
  }
```

```
  public static void InserirContato(String nome, String endereco, String
email, String empresa)
  {
    Contato novoContato = new Contato(nome, endereco, email, empresa);
    minhaLista.add(novoContato);
  }
  public static void ApagarContatoPorNome(String nome)
  {
    for (int i = 0; i < minhaLista.size(); i++)
    {
      String nomeVez = minhaLista.get(i).nome;
      if(nomeVez.equals(nome))
      {
        minhaLista.remove(i);
        System.out.println("Nome Removido: " + nome);
        break;
      }
    }
  }

 public static void ImprimirContato(Contato meuContato)
  {
    if(!meuContato.nome.equals("n/a"))
    {
      System.out.println(meuContato.nome + "\nEmpresa: " +
meuContato.empresa + "\nemail: " + meuContato.email + "\nTelefone: " +
meuContato.telefone);
    } else {
      System.out.println("CONTATO N/A!");
    }
  }

  public static void ImprimirLista()
  {
    System.out.println("LISTA COMPLETA:");
    for(Contato contato:minhaLista)
    {
      System.out.println(contato.nome);
    }
  }
}
```

DESAFIO FINAL

Uruca

Agora todo o conhecimento que obtivemos durante este Livro será aplicado na construção deste programa.

Escreva e Compile linha por linha, método por método, até entender todos os conceitos e idéias por trás da construção de toda esta aplicação de cadastro de contatos.

42. Banco de Dados

O que criamos anteriormente, é uma tabela relacional de valores. Pense que toda a estrutura de dados de um programa ou *site* pode estar contida em uma tabela desta.

Além de gravar nomes e dados de usuários, podemos salvar as preferências do usuário durante a navegação em um *site* ou aplicativo.

É possível, portanto, ordenar dados da partida de um jogo, atributos da construção de objetos visuais ou gráficos.

Os Bancos de Dados mais modernos, são ferramentas que, além de adicionar e remover itens em tabelas relacionais, oferecem-nos métodos mais sofisticados para organizar e ler estas tabelas.

NOME	TELEFONE	EMAIL	EMPRESA
Bugado	555-6666	bugado@mail.com	ACME Inc.
Felpudo	1234	contato@mail.com	Escola do VideoGame
Tito Petri	555-1234	tito.petri@gmail.com	Tito Coorp.
Fofura	1234	contato@mail.com	Escola do VideoGame
Lesmo	999-8888	lesmo@mail.com	ACME Inc.
Uruca	555-6666	uruca@mail.com	ACME Inc.

A tabela anterior está totalmente desordenada. Imagine se você pudesse ordená-la por cada um dos atributos, veja quanta coisa interessante poderia descobrir.

Abaixo podemos ordenar todos os itens do Banco, de acordo com a ordem alfabética do campo NOME por exemplo.

NOME	TELEFONE	EMAIL	EMPRESA
Bugado	555-6666	bugado@mail.com	ACME Inc.
Felpudo	1234	contato@mail.com	Escola do VideoGame
Fofura	1234	contato@mail.com	Escola do VideoGame
Lesmo	999-8888	lesmo@mail.com	ACME Inc.
Tito Petri	555-1234	tito.petri@gmail.com	Tito Coorp.
Uruca	555-6666	uruca@mail.com	ACME Inc.

Ou agora, ordenar pelo número de telefone. Só de visualizar a tabela, conseguimos entender quais são os usuários que possuem o mesmo telefone, por exemplo.

NOME	TELEFONE	EMAIL	EMPRESA
Felpudo	1234	contato@mail.com	Escola do VideoGame
Fofura	1234	contato@mail.com	Escola do VideoGame
Tito Petri	555-1234	tito.petri@gmail.com	Tito Coorp.
Bugado	555-6666	bugado@mail.com	ACME Inc.
Uruca	555-6666	uruca@mail.com	ACME Inc.
Lesmo	999-8888	lesmo@mail.com	ACME Inc.

E assim, podemos enxergar nossa lista por várias ordenações diferentes, como o email:

NOME	TELEFONE	EMAIL	EMPRESA
Bugado	555-6666	bugado@mail.com	ACME Inc.
Felpudo	1234	contato@mail.com	Escola do VideoGame
Fofura	1234	contato@mail.com	Escola do VideoGame
Lesmo	999-8888	lesmo@mail.com	ACME Inc.
Tito Petri	555-1234	tito.petri@gmail.com	Tito Coorp.
Uruca	555-6666	uruca@mail.com	ACME Inc.

E a empresa, por exemplo:

NOME	TELEFONE	EMAIL	EMPRESA
Bugado	555-6666	bugado@mail.com	ACME Inc.
Lesmo	999-8888	lesmo@mail.com	ACME Inc.
Uruca	555-6666	uruca@mail.com	ACME Inc.
Felpudo	1234	contato@mail.com	Escola do VideoGame
Fofura	1234	contato@mail.com	Escola do VideoGame
Tito Petri	555-1234	tito.petri@gmail.com	Tito Coorp.

Se você fosse programar a listagem de itens de um *site* ou aplicativo, perceba agora como poderia ser útil este tipo de recursos.

Hoje em dia existem muitas ferramentas de **Banco de Dados**. Entre elas, posso sugerir duas muito úteis e famosas:

SQL é uma linguagem de programação específica para **Bancos de Dados**. Muitas linguagens de programação conversam com **SQL**, e poderiam se integrar facilmente a um **Banco de Dados** em **SQL,** e utilizá-lo em *sites* e aplicativos *iOS* ou *Android*, por exemplo.

Firebase é o que chamamos de **BackEnd de Dados**. O **Firebase** é um *site*/serviço onde você pode criar e consultar **Bancos de Dados** com muita facilidade.

Como ainda estamos dando nossos primeiros passos em ***Java*** e utilizando um compilador um pouco limitado em recursos, não existe uma integração simples com o **SQL**.

Porém, quando você der os próximos passos e começar a programar seus aplicativos para o sistema ***Android*** por exemplo (**utilizando o *IDE Android Studio***), você já terá entendido todos os conceitos de base necessários para se trabalhar com o ***Java***, e as ideias principais sobre **Bancos de Dados**.

43. Armazenamento de Dados

Quando criamos uma variável, ela fica armazenada temporariamente na memória do computador. A partir do momento em que você fechar o *browser* ou a janela do compilador, esse dado se apaga.

Porém, existem maneiras de armazenar os dados permanentemente na memória. É o que chamamos de **Dados Persistentes**. Perceba que quando seu computador ou celular é desligado e ligado novamente, todos os dados (fotos, vídeos, arquivos) ainda permanecem lá.

Isto é porque eles foram armazenados permanentemente em uma memória sólida ou rígida - *Hard Disk*. Um tipo de memória em que o dado fica gravado para sempre.

Diferente da memória RAM, onde todos os dados são zerados quando a máquina desliga.

Podemos armazenar os dados permanentemente de algumas maneiras diferentes:

Gravando na memória sólida (*HD* ou disco rígido). Todo equipamento possui uma memória, pois é lá que está

instalado o sistema operacional, ou o sistema principal da máquina *(Windows, Mac OS, Android ou iOS*).

Gravando em algum cartão de armazenamento, como o do seu celular, um *pendrive* ou um HD externo ao seu computador.

Gravando o dado na nuvem. Como hoje a conexão com a internet já é quase que constante, muitas aplicações gravam o dado direto em algum computador/servidor que esteja conectado através da *internet*.

Ou seja, o seu aplicativo se conecta ao servidor e manda o dado para ser armazenado em algum computador específico (servidor).

Aqui no compilador *online* de ***Java,*** também seria uma tarefa bem trabalhosa armazenar o **Dado** permanentemente na aplicação. Porém, quando for para ***IDEs*** como o *Android Studio*, vai perceber que esta tarefa é feita com muita facilidade.

44. Baixando e Executando o Projeto

No compilador que estou utilizando, é possível baixar o projeto no botão "**DOWNLOAD .ZIP"**.

Dentro da pasta compactada, você vai encontrar o seu código em texto, na extensão **.java** e o programa do Java compilado, em extensão **.jar**.

O **.jar** é basicamente um executável do Java que você pode executar se tiver o Java Runtime Environment instalado no computador (e geralmente qualquer computador de hoje em dia já tem o Java instalado. Caso você não tenha, baixe gratuitamente em https://www.java.com/en/download/

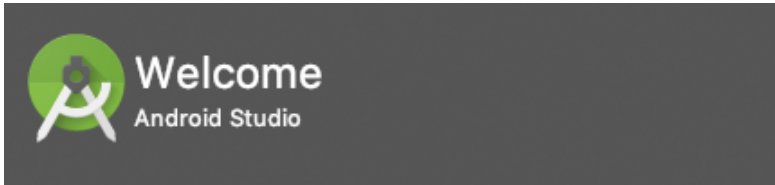
Com o Java Runtime instalado, basta você executar o **.jar** pelo Terminal (Mac OS) ou CMD (Windows) através das linhas de comando:

```
java -jar /Users/apple/Downloads/MeusProgramas/MeuApp.jar {"Tito", "Felpudo", "Lesmo", "Bugado", "Uruca"}
```

Perceba que existe **4 argumentos** na linha de execução:

java (para executar um programa com o JRE)
-jar (extensão que será executada)
caminho+nome (diretório onde se encontra o arquivo **.jar**)
Argumento (caso a aplicação espere por algum. Lembra do *String[ ] args ?*)

Android Studio 3.2.1
Instalação e Primeiros Passos
© Tito Petri - 20 - Todos os Direitos Reservados.
Java
Tito Petri
Professor e Desenvolvedor
www.titopetri.com

44. Android Studio

O Android Studio é uma ferramenta criada pelo Google, mesma empresa responsável pela criação e desenvolvimento do sistema Android.

Ele nos possibilita criar aplicações para qualquer tipo de dispositivo que funcione com o sistema Android como **Celulares**, **Tablets**, **Relógios** (*AndroidWear*) e até mesmo o **Google Glass** (*SmartGlass*) e os futuros **Automóveis Auto-Dirigidos** (*Google Self-Driving Car*).

No Android Studio, utiliza-se a Linguagem Java para programar as aplicações e agora podemos acessar e controlar todos os recursos dos dispositivos Android para utilizar nas nossas programações:

Para programar aplicativos é indispensável conhecer todo este caminho até aqui, saber o BÊ-A-BÁ da programação.

Tudo o que aprendemos neste Livro será utilizado 100% do tempo enquanto você estiver trabalhando com o Android Studio ou qualquer outro IDE que construa softwares ou aplicativos para celulares e computadores.

Recentemente, o Google integrou uma nova linguagem ao Android Studio, o **Kotlin**. Uma linguagem com sintaxe mais simplificada e resumida que o Java.

No entanto, ainda aconselho aos alunos a aprender muito bem o Java antes de partir para qualquer outra linguagem.

Ainda existem milhares de projetos no mundo, desenvolvidos em Java.

A linguagem Java é muito utilizada ainda, e mesmo que ele venha a não existir mais algum dia, esta transição ainda demoraria muitos anos para acontecer totalmente.

Fora que o Java tem tanta regra e especificação, que no final, quem aprende o Java corretamente vai entender qualquer outra linguagem de programação (**Kotlin**, **Swift**, **Python** ou **C#**) com muito mais facilidade.

O Java possui regras e boas práticas que são comuns a qualquer linguagem.

Se o programador entende bem de Java, qualquer outra linguagem vai ser moleza.

a) Download e Instalação

Instalar o Android Studio é compilar a sua primeira aplicação, é a parte mais chata e trabalhosa do processo.

Os pacotes são grandes e podem demorar para serem baixados e instalados, por isso seja muito paciente na hora de fazer este processo.

Também sugiro que você tenha um bom computador para estar utilizando o programa.

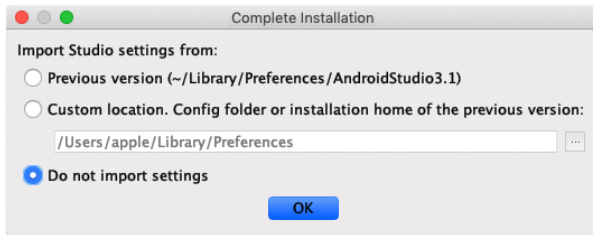
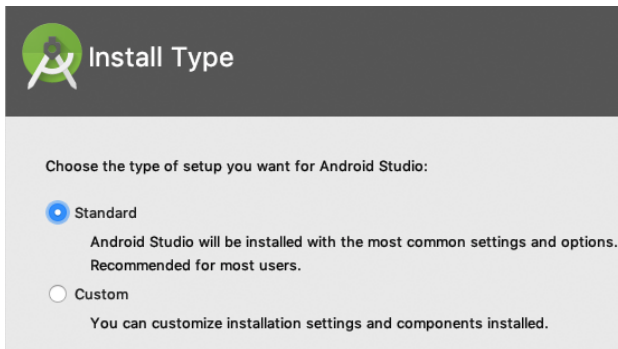
Trabalho com um processador i5 e 8gb de memória RAM, sugiro um computador equivalente ou próximo à isso.

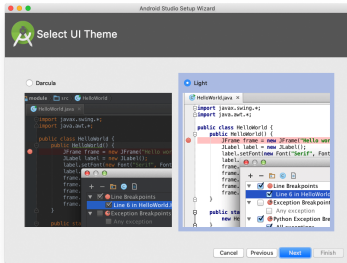
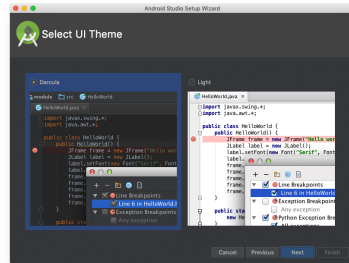
Não é necessário possuir um dispositivo Android pois existe a possibilidade de desenvolver e testar os aplicativos no simulador.

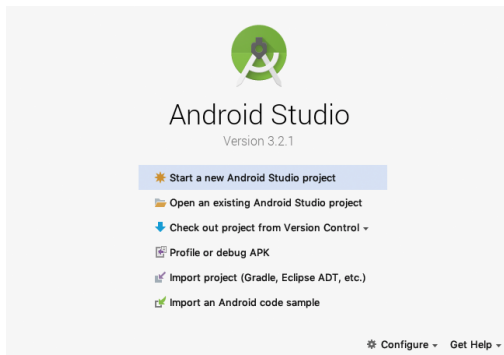
O primeiro passo é **baixar** e **executar** o instalador do Android Studio IDE através do link

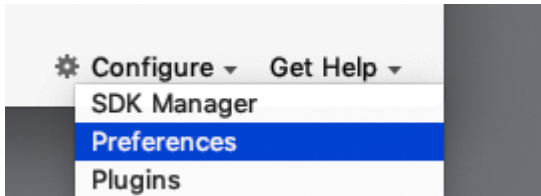
https://developer.android.com/studio/

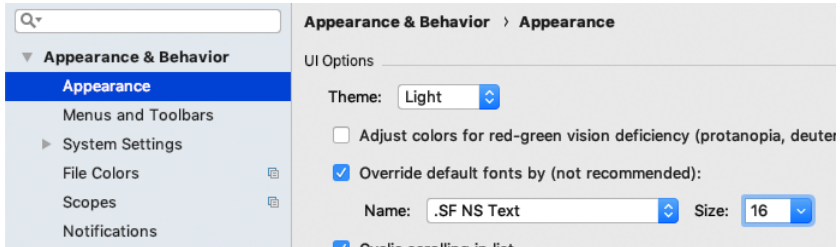
Esta opção pede sua configuração de preferências, caso você já tenha instalado o Android Studio alguma vez anteriormente, pode aproveitar suas configurações anteriores.

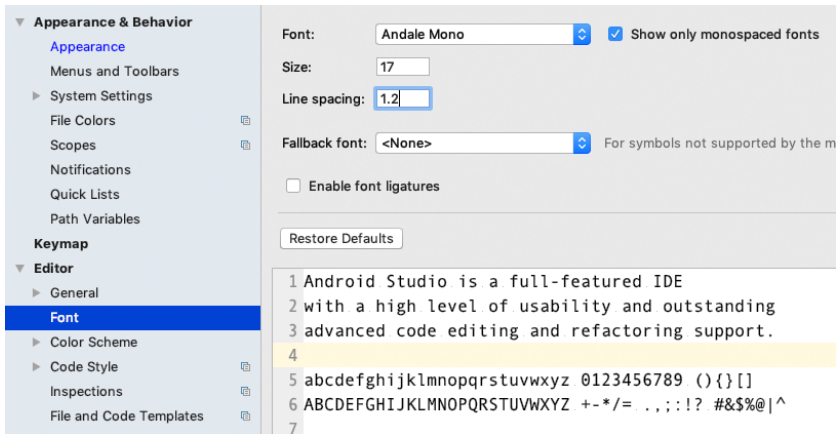
Como é nossa primeira instalação ainda, deixe marcado em ***Do not import settings***.

Prossiga com a instalação padrão (***Standard).***

b) Temas e Preferências

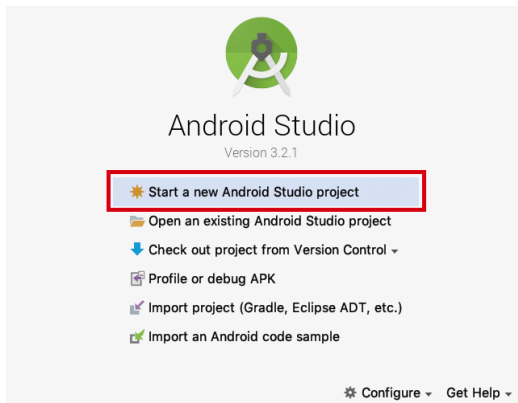
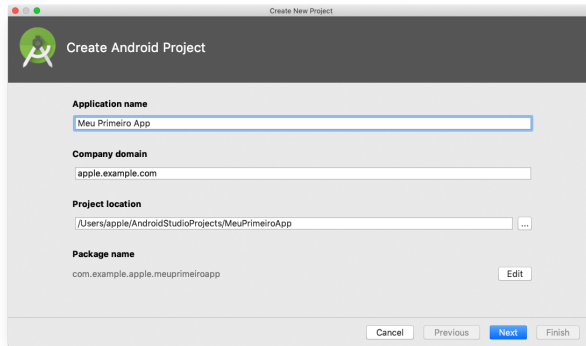
Aqui você deve escolher entre um dos 2 **Temas de Cor** das janelas e menus do programa. Claro (Light) e escuro (Darcula).

Light

Darcula

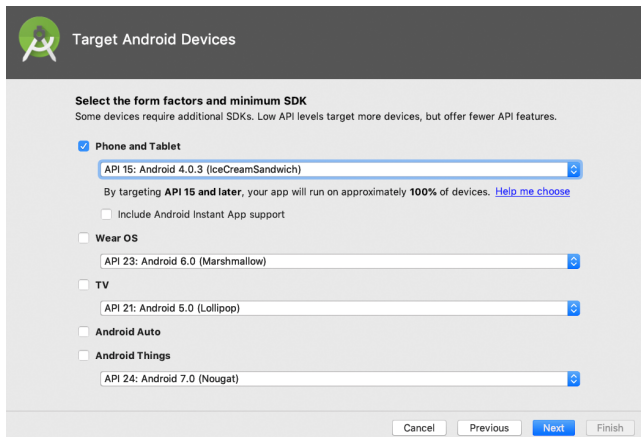
Ao terminar a instalação, aparecerá a seguinte tela:

Antes de criar nosso primeiro projeto, vamos aprender como configurar o **tamanho dos textos** no programa.

Acesse a opção *Configure - Preferences*

Em *Appearance - UI Option* pode-se alterar o tamanho dos textos da **interface do programa**.

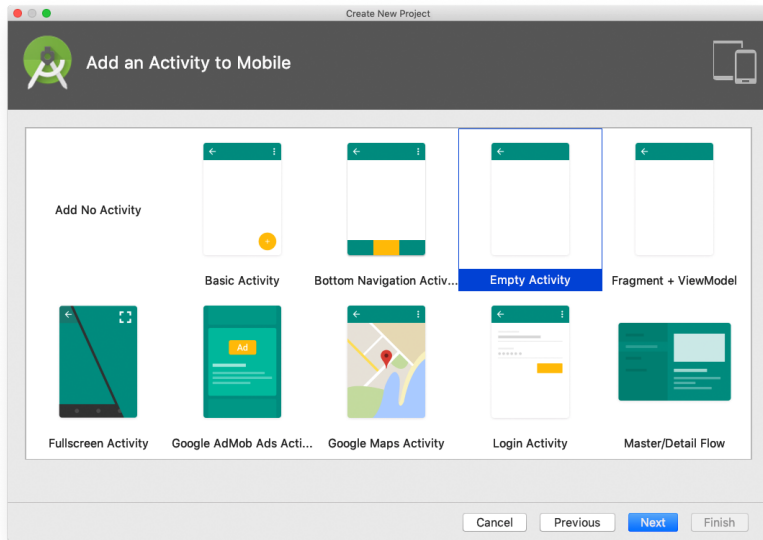
Ligue a opção **Override default fonts** e deixe em o **Size** em **16**.

** Aumentar o tamanho do texto da interface, pode fazer com que alguns textos fique fora da margem em algumas janelas do programa, por isso existe a mensagem de não recomendado (not recommended). Farei apenas com a finalidade de expor melhor os textos e nomes neste material.*

Nas opções Editor - Font podemos alterar o tamanho da fonte dos **códigos do projeto**. Deixe o **Size** em **17**.

c) Criando um Novo Projeto

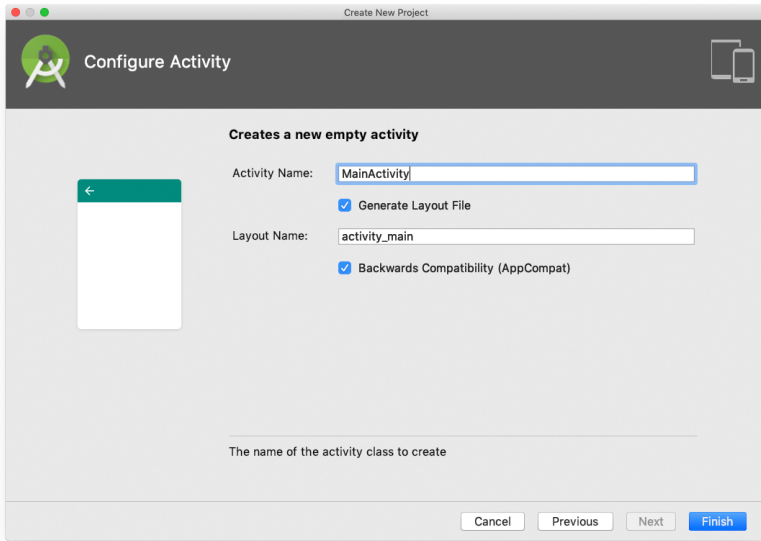
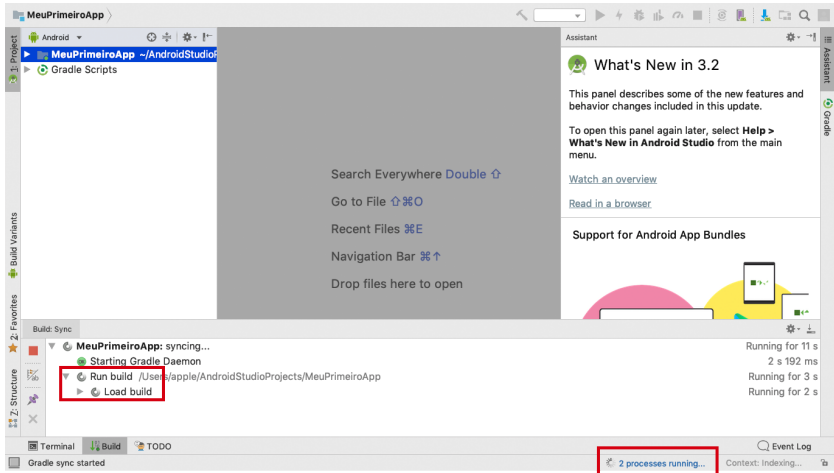
Agora sim crie seu primeiro projeto Android clicando na opção ***Start a new Android Studio project***.

Dê um nome do seu Aplicativo. Usarei o nome **Meu Primeiro App**.

Agora selecione a versão mínima requerida para executar o seu aplicativo. Vou usar **API 16: IceCream Sandwich**

Isto garante que aparelhos com versões mais antigas do sistema Android consigam executar a sua aplicação.

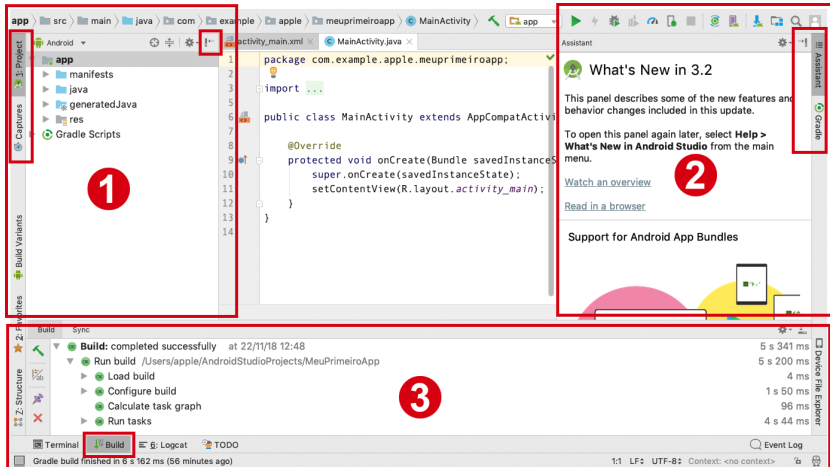
O próximo passo é selecionar um template inicial para sua aplicação.

Utilize o modelo em branco **Empty Activity**.

O próximo passo é definir o nome da classe da tela inicial da aplicação.

Este nome vem por padrão como MainActivity, deixarei este nome mesmo. O outro campo Layout Name serve

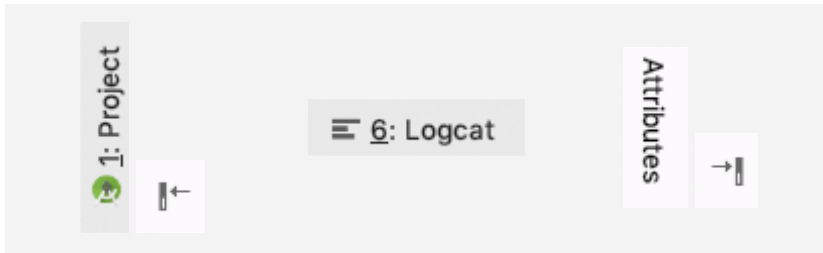
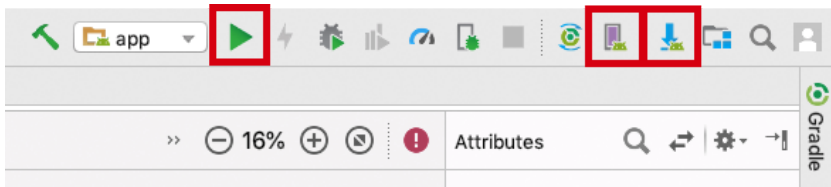
para definir o arquivo xml que vai montar o layout da nossa tela. Deixarei o nome padrão **activity_main**.

Siga com a criação do projeto até o final, e aguarde até que todos os arquivos e pastas do projeto sejam criados. Isto pode demorar até alguns minutos.

Observe as indicações de carregamento que irão aparecer na tela e espere todos os processos terminarem.

Quando o projeto acabar de ser criado e carregado, você vai enxergar uma tela assim.

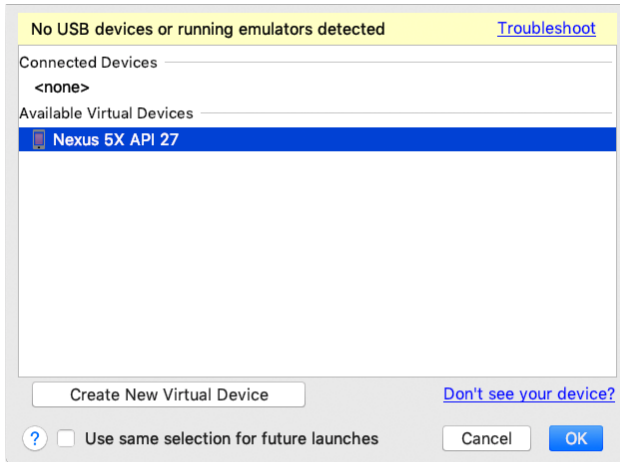
Podemos dividir a interface do Android Studio em basicamente 3 áreas:

1) **Project**, onde você vai ter um explorador para navegar entre os arquivos que formam o projeto;

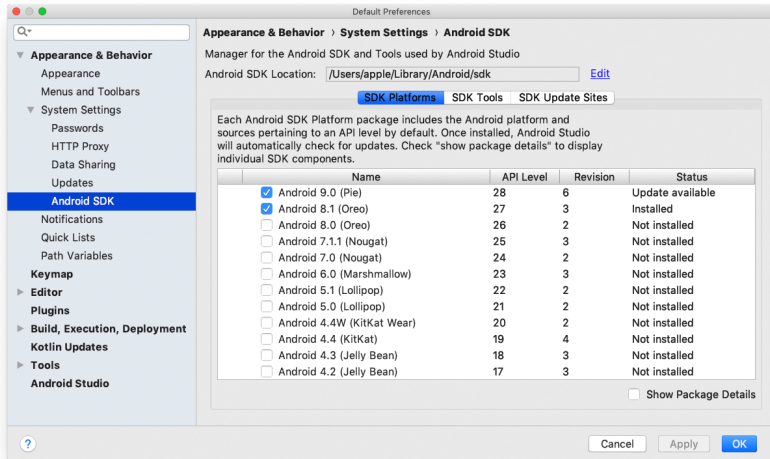
2) **Atributos** ou **Propriedades** dos objetos selecionados;

3) **Console** de Debug.

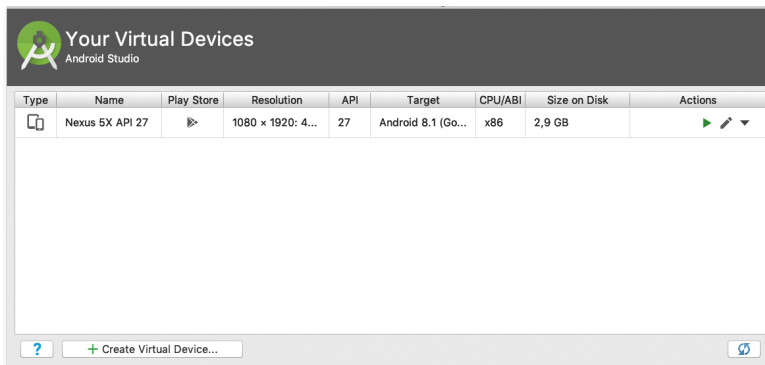
Clique nos seguintes ícones para expandir ou recolher cada uma das respectivas abas.

Preste atenção também nos seguintes comandos na barra superior: **Run AVD** e **SDK Manager**

d) Executando a Aplicação

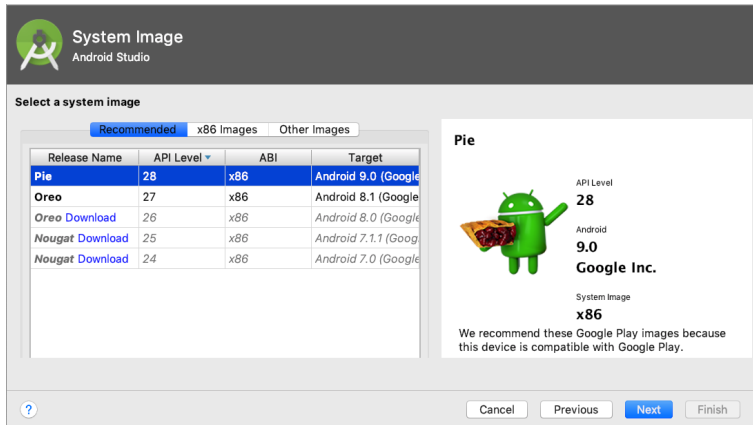
Para executar a aplicação clique no ícone do Play / Run.

Aqui vai aparecer a sua lista de simuladores criados (caso tenha algum) e a lista de aparelhos Android ligados na USB do seu computador. Selecione por onde deseja executar a aplicação.

e) Android SDK Manager

SDK Manager esta é a janela por onde controlamos o download dos pacotes e versões do android studio.

f) Simulador (Android Virtual Device)

Android Virtual Device é o simulador virtual.

Se você não possui um dispositivo android, pode criar e utilizar um simulador para rodar a sua aplicação.

Ao abrir o AVD você verá a lista de simuladores criados.

Se ainda não existir nenhum, clique em **Create Virtual Device** e crie o simulador da última versão do android (Oreo ou Pie).

Caso não tenha essas versões instaladas, faça o download.

g) Android Studio e Simulador AVD - Resolução de Problemas

1) Faça todas as atualizações do android studio
2) Baixe o ultimo SDK e versão do AVD(OREO)

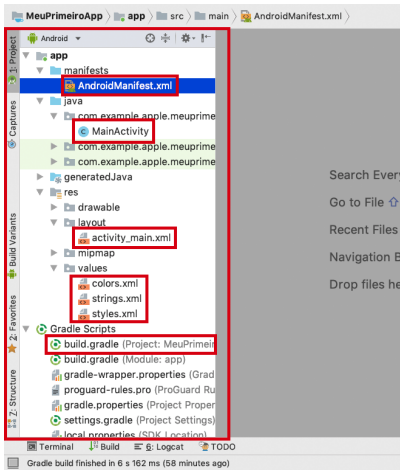
3) Tenha certeza que o intel HAXM esteja instalado
https://software.intel.com/en-us/articles/intel-hardware-accelerated-execution-manager-intel-haxm

4) abra o Android Studio (algumas vezes) e deixe ele carregando todos os processos
(observe na barra inferior que ele fica fazendo download de pacotes ou atualizando sempre algo)

5) A Configuração recomendada para rodar o Android Studio e o Simulador é um i5 com 8gb de RAM (ou equivalente)

* Caso possua uma configuração inferior, recomendo executar os aplicativos no seu próprio aparelho Android, isso vai economizar bastante memória se não utilizar o simulador.

g) Anatomia do Projeto Android

Um projeto de android é composto por centenas de arquivos. Precisaremos entender apenas alguns deles para começarmos a mexer no nosso aplicativo.

Navegue na aba project (3) e encontre os seguintes arquivos:

- AndroidManifest.xml
- MainActivity
- Activity_main.xml
- Colors / Strings.xml
- build.gradle

Cada um deles é responsável por alguma tarefa ou configuração do nosso app.

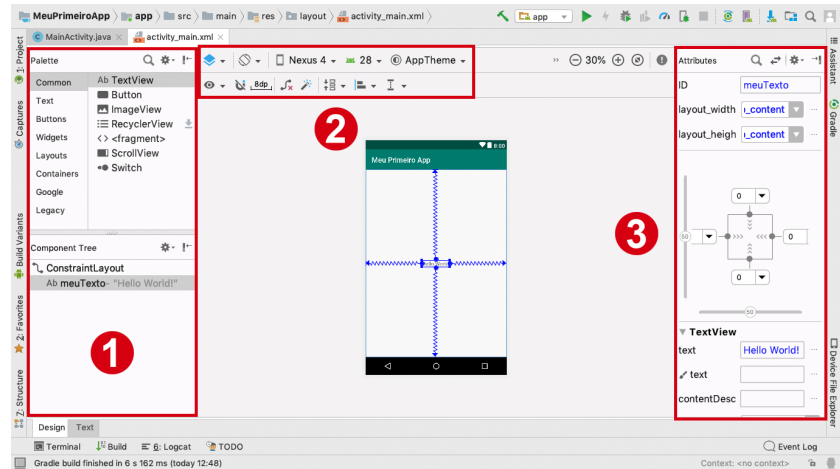
Por enquanto, apenas lembre-se que o seu **código** principal da aplicação é a **MainActivity**. É nesta classe que vamos começar a programação do nosso app.

Lembre-se tambeém do **activity_main** que define o *layout* da nossa primeira tela.

h) Componentes Nativos

Acessando o **activity_main.xml** você vai para o editor de interfaces. Aqui podemos criar e configurar os **componentes nativos do Android** como janelas, botões,

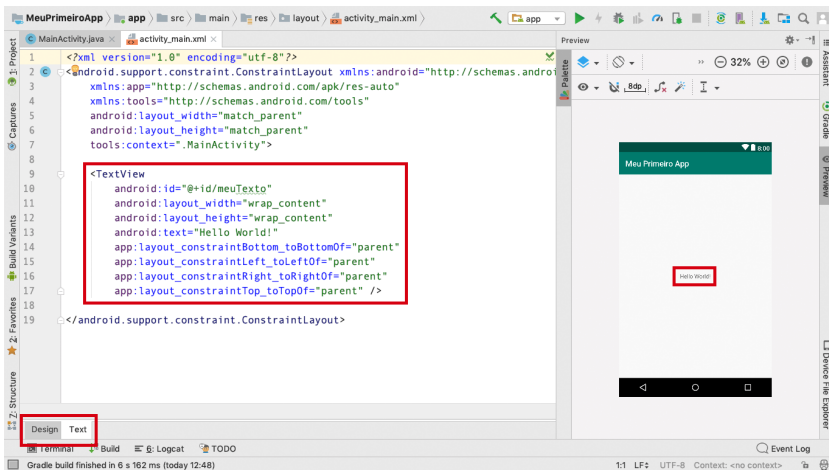
textos, imagens ou qualquer outro componente visual de interface.

Você pode dividir esta tela em 3 áreas:

1. Paleta de Componentes e Hierarquia da View (Tela)
2. Ferramentas de Visualização e Layout
3. Propriedades dos Componentes

Nos botões abaixo, **Design** e **Text** podemos alternar entre **Layout** e **Código** .xml.

Perceba que toda a montagem no nosso layout é baseada em uma estrutura XML. Cada componente (botão, texto ou imagem) é definido através de *tags* que são montadas neste arquivo de texto enquanto estamos trabalhando com o editor.

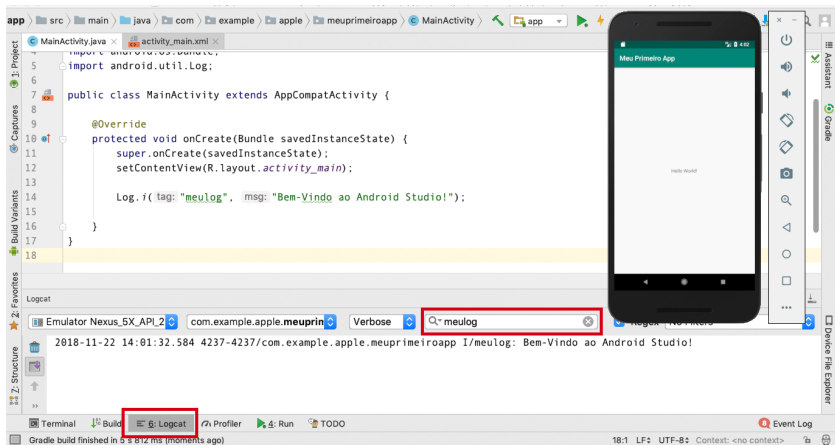
Geralmente utilizamos o **Editor** para montar o nosso *layout* porém é interessante você observar e entender também como funcionam as *tags* do XML

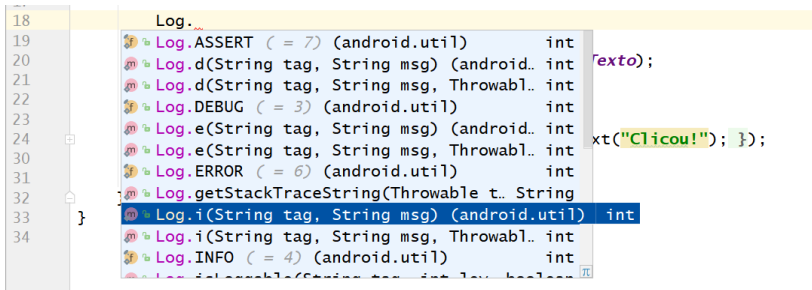
.

f) Hello World

Vamos agora escrever nosso primeiro trecho de código em Java dentro de uma aplicação Android.

Acesse o arquivo **MainActivity.java**. Esta é a classe que controla a primeira tela do nosso app.

Perceba que ela estende a classe **AppCompatActivity**, então herda todas suas propriedades e métodos.

Insira um comando Log dentro da função on create.
O Log equivale ao comando System.out.print() que utilizamos anteriormente no compilador online.

Perceba que ao começar a digitar **Log**, uma janela vai se abrir com algumas sugestões de comando.

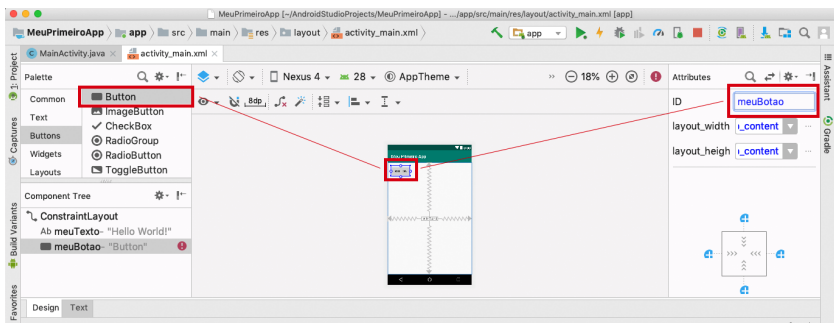
Selecione o **Log.i** e aperte o **Enter**.

Perceba que o comando já vai ser completado!

```
Log.i(tag:meulog, msg:"Bem Vindo ao Android Studio");
```

E se você observar logo no início do código, a biblioteca do Log foi importada também. (*import android.util.Log;*)

O Console do Android Studio exibe MUITAS mensagens de status e execução. Para isso existe a **tag**, ela vai nos ajudar a filtrar as mensagens do console e exibir apenas mensagens assinadas com a nossa **tag**.

Para isso, entre com o mesmo nome que você usou em **tag**, no campo de busca (ícone da lupa) nas opções do Console.

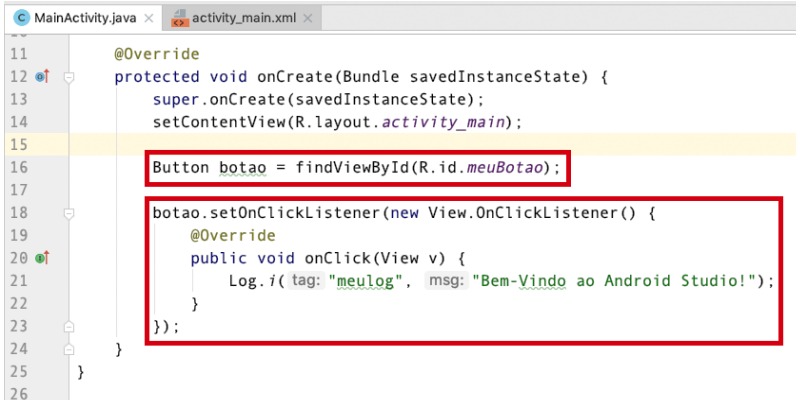
i) Identificando os Componentes

Cada componente no nosso *layout* possui uma identificação (ID).

Selecione algum componente como um botão por exemplo, e observe a aba de **atributos** do editor de layouts. Insira algum nome para identificá-lo.
Cada ID que associamos a um componente, fica registrado em um arquivo na **raíz do projeto do Android**, (R.id. …).

Usaremos este identificador posteriormente para manipular este componente no nosso código em Java.

Para o Android Studio, o componente visual ***Button***, não passa de um objeto criado a partir de uma classe, no caso a própria classe ***Button***.

Agora vamos para o arquivo ***MainActivity.java***, é lá onde vamos inserir nosso código em Java para começar a programar a interatividade do aplicativo.

Primeiro, precisamos criar um novo objeto na programação, este objeto vai ser do tipo ***Button***. É como declarar uma variável, do tipo ***Button***. Utilizamos:

```
Button botao = findViewById(R.id.meuBotao);
```

Onde ***Button*** é a classe/tipo do objeto, e ***botao*** é o nome da minha variável.

O comando **findViewById** vai procurar no layout, o objeto que tem o id associado.

j) Evento de Clique no Botão

Agora que já identificamos o botão no nosso código, vamos criar um evento de **Clique no Botão.**

MainActivity.java × activity_main.xml ×

```
11      @Override
12      protected void onCreate(Bundle savedInstanceState) {
13          super.onCreate(savedInstanceState);
14          setContentView(R.layout.activity_main);
15
16          Button botao = findViewById(R.id.meuBotao);
17
18          botao.setOnClickListener(new View.OnClickListener() {
19              @Override
20              public void onClick(View v) {
21                  Log.i( tag: "meulog", msg: "Bem-Vindo ao Android Studio!");
22              }
23          });
24      }
25  }
26
```

Digite cuidadosamente o código acima, perceba que o **TAB** autocompleta as expressões para você.

Dentro do método ***OnClickListener*** você pode inserir um **Log** apenas para testar.

A mensagem do ***Log*** deve ser exibida agora cada vez que clicarmos no botão.

k) Mudando o texto do TextView

Agora vamos fazer uma interação simples. Crie um componente de texto ***TextView*** no seu layout, identifique-o e declare no código, exatamente da mesma forma como fizemos com o componente ***Button***.

MainActivity.java activity_main.xml

```
12        @Override
13        protected void onCreate(Bundle savedInstanceState) {
14            super.onCreate(savedInstanceState);
15            setContentView(R.layout.activity_main);
16
17            Button botao = findViewById(R.id.meuBotao);
18            TextView texto = findViewById(R.id.meuTexto);
19
20            botao.setOnClickListener(new View.OnClickListener() {
21                @Override
22                public void onClick(View v) {
23                    texto.setText("Viva o Android Studio!");
```

'ariable 'texto' is accessed from within inner class, needs to be declared final

```
26        }
27    }
```

Agora dentro do evento do clique.

45. Linguagens de Programação

Existem muitas ferramentas e linguagens de programação diferentes.

Frequentemente os alunos me perguntam qual é a melhor ferramenta ou qual linguagem aprender?

Sugiro que esta pergunta seja sempre trocada para:

"***Que tipo de aplicação você gostaria de desenvolver?***"

Baseado nesta pergunta, escolha uma ferramenta ou linguagem que vai te dar os recursos para desenvolver este seu produto. Veja as principais ferramentas e linguagens utilizadas para diferentes tipos de desenvolvimento:

Páginas, Web Sites e Sistemas para Internet:
HTML
CSS
Php
JavaScript

** Lembrando que o HTML e CSS teoricamente não são linguagens de programação, e sim de marcação, pois servem apenas para montar e configurar as páginas, e não para montar algoritmos. Porém essas linguagens e conhecimentos devem estar presentes no dia-a-dia do Desenvolvedor Web.*

Aplicativos Mobile e Desktop, recomendo as seguintes linguagens e IDEs.

Desktop & Mobile Apps	IDE	Plataforma
Java	Android Studio	Android
Swift	Xcode	Mac OS e iOS
C-Sharp (C#)	Visual Studio	Windows

E no caso do desenvolvimento de Games, Realidade Virtual e Realidade Aumentada, recomendo as seguintes ferramentas:

Games e VR	IDE	Plataforma
Swift	Xcode	Mac OS e iOS
C-Sharp (C#)	Unity	Multiplataformas

Não existe uma linguagem ou ferramenta melhor ou pior, o principal é entender as diferenças e conhecer qual delas vai te ajudar a chegar no produto que você deseja criar.

Lembre-se que o primeiro passo sempre será *"Entender qual é o produto que você deseja saber criar!"*

…

Parabéns querido aluno por chegar até aqui e ter adquirido mais este valioso conhecimento!

Se quiser aprender sempre mais sobre criação de Jogos e Aplicativos, não deixe de conhecer o Aprenda Programar, meu portal de cursos online onde você pode se especializar em:

- Algoritmos e Lógica de Programação
- Modelagem e Animação 3D
- Criação de Personagens para Jogos e Filmes
- Programação de Aplicativos Nativos para iOS e Android
- Criação de Games 2D, 3D e Realidade Virtual
- Realidade Aumentada e Visão Computacional
- Metodologia STEAM
- Robótica e Impressão 3D

Para virar aluno do Aprenda Programar você deve se inscrever pela plataforma Hotmart, no link abaixo.

Adquira seu acesso para sempre ao Aprenda Programar:
https://hotmart.com/product/en/aprenda-programar-com-tito-petri